

# THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

November 04, 2000

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE UNDER 35 USC 111.

APPLICATION NUMBER: 60/145,426

FILING DATE: July 23, 1999

## PRIORITY DOCUMENT

SUBMITTED OR TRANSMITTED IN  
COMPLIANCE WITH RULE 17.1(a) OR (b)

BEST AVAILABLE COPY



By Authority of the  
COMMISSIONER OF PATENTS AND TRADEMARKS

*W. Montgomery*  
W. MONTGOMERY  
Certifying Officer

**PROVISIONAL APPLICATION FOR PATENT  
COVER SHEET**

Case No. VANM121.001PRF  
Date: July 23, 1999  
Page 1

ASSISTANT COMMISSIONER FOR PATENTS  
WASHINGTON, D.C. 20231

ATTENTION: PROVISIONAL PATENT APPLICATION

Sir:

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR § 1.53(c).

For: **METHOD AND APPARATUS FOR HIGH-SPEED SOFTWARE RECONFIGURABLE  
CODE DIVISION MULTIPLE ACCESS COMMUNICATION**

Name of First Inventor: Nico Lugil  
Residence Address: Vrouwenparklaan 31, B-3110 ROTSELAAR, BELGIUM

Name of Second Inventor: Eric Borghs  
Residence Address: Kollegestraat 75, B-2440 GEEL, BELGIUM

Name of Third Inventor: Sébastien Louveaux  
Residence Address: Avenue de l'Equerre 25 B302, B-1348 LOUVAIN-LA-NEUVE,  
BELGIUM

Name of Fourth Inventor: Carl Mertens  
Residence Address: Het Venneke 2, B-2930 BRASSCHAAT, BELGIUM

Name of Fifth Inventor: Lieven Philips  
Residence Address: Kleine Kruisweg 9A, B-3201 AARSCHOT, BELGIUM

Name of Sixth Inventor: Jurgen Vandermot  
Residence Address: Diestsestraat 250 B3, B-3000 LEUVEN, BELGIUM

Name of Seventh Inventor: Jan Vanhoof  
Residence Address: Wijgmaalbroeck 59, B-3018 WIJGMAAL, BELGIUM

Enclosed are:

- (X) Specification in 191 pages.
- (X) A check in the amount of \$150 to cover the filing fee is enclosed.
- (X) A return prepaid postcard.
- (X) The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Account No. 11-1410. A duplicate copy of this sheet is enclosed.

PROVISIONAL APPLICATION FOR PATENT  
COVER SHEET

Case 1 VANM121.001PRF

Date: July 23, 1999

Page 2

Was this invention made by an agency of the United States Government or under a contract with an agency of the United States Government?

(X) No.

( ) Yes. The name of the U.S. Government agency and the Government contract number are:

(X) Please send correspondence to:

John M. Carson  
Knobbe, Martens, Olson & Bear, LLP  
620 Newport Center Dr., 16th Floor  
Newport Beach, CA 92660

Respectfully submitted,

John M. Carson  
Registration No. 34,303

S:\DOCS\UMC\MC-2958.DOC  
072399

60145426-072399

KNOBBE, MARTENS, OLSON & BEAR, LLP  
620 NEWPORT CENTER DR. 16TH FLOOR NEWPORT BEACH, CA 92660  
(949) 760-0404 FAX (949) 760-9502

# CDMA<sub>x</sub>

ARM Subsystem

Status: Preliminary

Doc no SC015001000

Last Update : July 12, 1999



## Contents

<b>1</b>	<b>ARM7TDMI interface and peripherals</b>	<b>5</b>
1.1	Boot-loader	7
1.1.1	Multi-CDMAx boot configuration	7
1.1.2	Boot-loader configuration with SP0 link	9
1.1.3	Boot-loader packet format	10
1.2	SPI-Controller	12
1.2.1	SPI inband commands	12
1.2.2	Events	13
1.2.3	Registers	13
1.2.4	Timing	15
1.3	Serial Port 0	16
1.3.1	Events	17
1.3.2	SP0 registers	17
1.3.3	Handshake Mode	20
1.3.4	Multi-CDMAx mode	21
1.4	General Purpose Input Output	25
1.4.1	GPIO registers	25
1.5	PWM-Timer	27
1.5.1	PWM-Timer registers	28
1.6	(S)UART driver	30
1.6.1	Flow-control	31
1.6.2	(S)Uart events	31
1.6.3	Asynchronous Timing	32
1.6.4	Synchronous Timing	33
1.6.5	Auto Bauding	34
1.6.6	(S)UART registers	35
1.7	Memory Managment Unit	39
1.7.1	External signal list	40
1.7.2	Internal signal list	41
1.7.3	External Memory Mapping	41
1.7.4	MMU Registers	43
1.7.5	External Timing	44
1.8	ARM-Interface	48
1.8.1	Internal interface	48
1.9	Interrupt Controller	51
1.9.1	Interrupt Controller registers	52
1.9.2	Interrupt Controller principle	53
1.10	Watchdog	54
<b>2</b>	<b>Memory map</b>	<b>55</b>
2.1	Normal operation	55
<b>3</b>	<b>ARM7TDMI timings</b>	<b>56</b>
3.1	ARM7TDMI timing	56
<b>4</b>	<b>Internal bus concept</b>	<b>57</b>
<b>5</b>	<b>Testability</b>	<b>58</b>
5.1	ATPG test	58
5.2	ARM debugger	60
5.3	Test enable register	60
<b>6</b>	<b>Gate-Count Components</b>	<b>60</b>

7 Tools	61
7.1 C-Code generation and simulation	61
7.1.1 Noboot mode program	61
7.1.2 Boot mode program	62
A Boot.s	64
B Regioninit.s	66
C Vectors.s	68
D Chandler.c	69
E Init.s	69

000000-02454709

## List of Figures

1	ARM7TDMI & Peripheral blockdiagram . . . . .	6
2	Multi-CDMAx Configuration . . . . .	8
3	Boot via SP0 link . . . . .	9
4	SP0 Parallel link principle . . . . .	10
5	Boot packet format . . . . .	11
6	Debug packet format . . . . .	11
7	Serial Peripheral Interface Controller . . . . .	12
8	Data transmit register . . . . .	13
9	Data receive register . . . . .	14
10	Configuration register . . . . .	14
11	CS and SCK configurable timings . . . . .	14
12	SPI timing . . . . .	15
13	SP0 block diagram . . . . .	16
14	SP0 transmit data register . . . . .	17
15	SP0 receive data register . . . . .	17
16	SP0 timer register . . . . .	18
17	SP0 control-status register . . . . .	18
18	SP0 handshake principle . . . . .	20
19	Ack-Request handshake principle . . . . .	20
20	SP0 multi-CDMAx boot directions . . . . .	21
21	SP0 boot example . . . . .	22
22	SP0 multi-CDMAx debug chain configuration . . . . .	22
23	SP0 debug example A . . . . .	23
24	SP0 debug example B . . . . .	23
25	GPIO blockdiagram . . . . .	25
26	Data GPIO control register . . . . .	25
27	Event GPIO control register . . . . .	26
28	PWM-Timer blockdiagram . . . . .	27
29	PWM-Timer synchronization . . . . .	28
30	Control register . . . . .	28
31	Period register . . . . .	29
32	Init register . . . . .	29
33	(S)UART blockdiagram . . . . .	30
34	Event Mask principle . . . . .	31
35	Event timings . . . . .	32
36	Asynchronous timings . . . . .	33
37	Synchronous timings . . . . .	33
38	Control register . . . . .	35
39	Mask register . . . . .	35
40	Transmit Data register . . . . .	36
41	Receive Data register . . . . .	36
42	Baud Rate register . . . . .	37
43	Status register . . . . .	38
44	MMU blockdiagram . . . . .	39
45	Memory connections for 4 external devices . . . . .	41
46	MMU-CSS Register . . . . .	43
47	MMU-GC Register . . . . .	44
48	Output Enable wait timing . . . . .	44
49	Data Enable wait timing . . . . .	46
50	External MMU timing for x wait states access . . . . .	47
51	External MMU timing with nEWAIT . . . . .	47
52	Interrupt controller blockdiagram . . . . .	51

53	Mask register . . . . .	52
54	Result register . . . . .	52
55	Branch register . . . . .	53
56	Interrupt principle . . . . .	54
57	Internal-External memory expand mode . . . . .	56
58	ARM7TDMI timing: APE = 0 . . . . .	57
59	ARM7TDMI timing: APE = 1 . . . . .	58
60	Internal data read bus concept . . . . .	59
61	Internal bus timing . . . . .	59
62	Peripheral scanpath . . . . .	60
63	JTAG debug interface . . . . .	60
64	Test control interface . . . . .	61

60445426-022209



## 1 ARM7TDMI interface and peripherals

### Features:

- ARM7TDMI core at 40 MHz.
- 2K x 32 SRAM, 0 WS access.
- CDMAx configuration via bootloader. Data from SPI or SP0 or EEPROM device or via the CDMA Development Accelerator interface or (S)UART interface.
- SPI controller with a maximum of 64Kx8 data range.
- Serial Port interface (SP0) with frame structure and handshake mechanism.
- PWM-Timer module. PWM puls generation or 32bit timer controller or puls width measurement.
- (S)UART driver with flowcontrol and flexible baudrate with auto bauding.
- Memory Managment Unit (MMU) with flexible external buswidth. Four chip select signals with a address range of 24 bits. Waitstate generator and external WAIT control.

60145425-072299

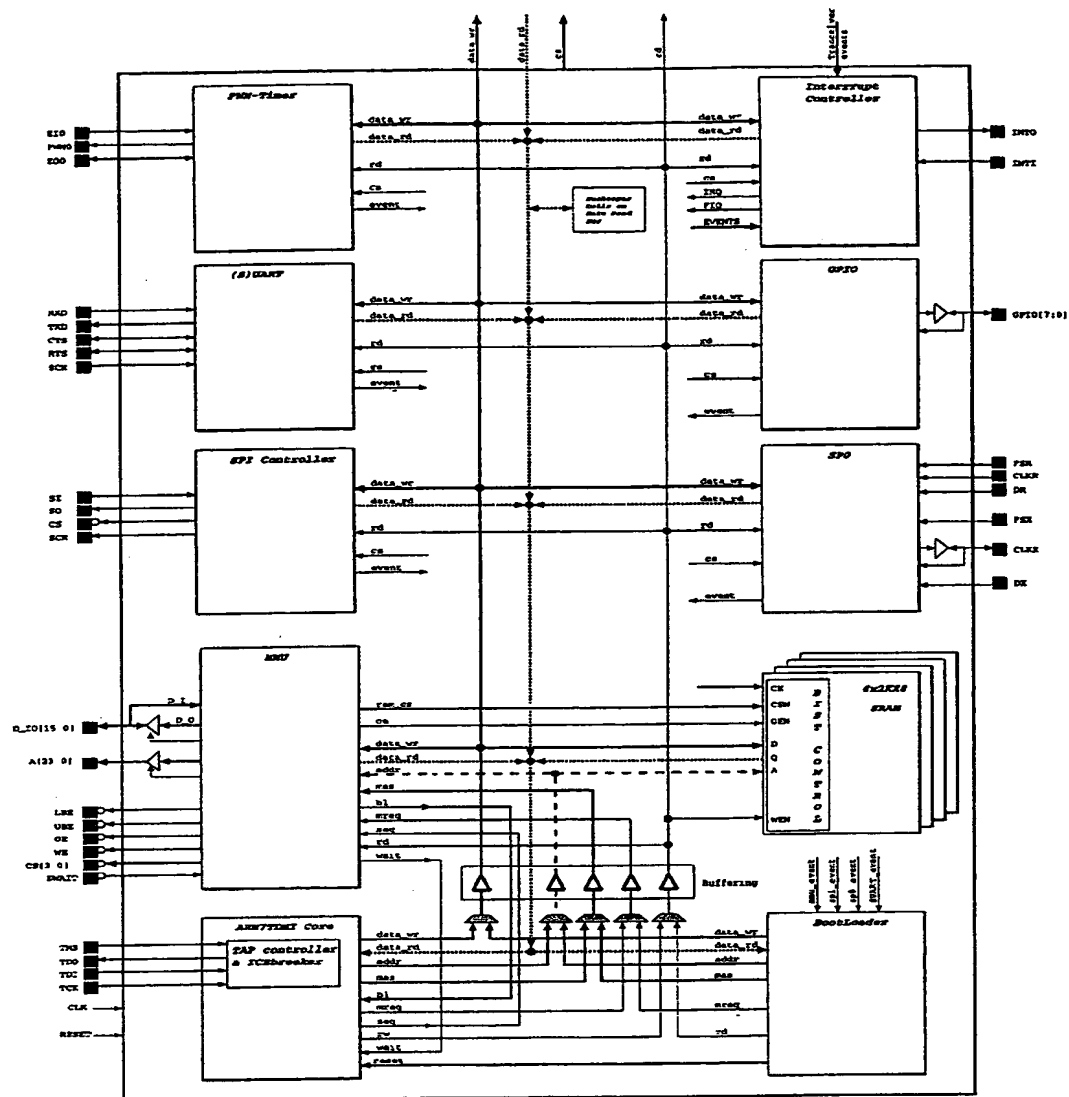


Figure 1: ARM7TDMI &amp; Peripheral blockdiagram

## 1.1 Boot-loader

The Boot-loader supports :

- Boot from SPI-device as **MASTER** with a variable boot length. Transfer of data to the SP0 interface in multi-CDMAx environment.
- Boot from external device on CS(0) in the memory map, through MMU as **MASTER** with a variable boot length. Transfer of data to the SP0 interface in multi-CDMAx environment.
- Boot from UART interface as **MASTER** with a variable boot length and auto bauding for rates between the 4K8 and 115K2. Transfer of data to the SP0 interface in multi-CDMAx environment.
- Boot from SP0 interface as **SLAVE** with a variable boot length and route to output of the SP0 interface.
- Multi-CDMAx boot operation possible with **MASTER-SLAVE** configuration.
- Write Boot data to internal/external SRAM/registers.
- ID-capture for multi-CDMAx configuration when using chain configuration. The first packet received by each CDMAx-ASIC indicates the ID and boot data. The next received packets are boot data and are executed by the ASIC when the ID-codes are identical. The boot operation for a CDMAx-ASIC is finished when a packet is received with length equal to zero.
- When booting from SPI or SP0 link, clock frequency between two CDMAx-chips is fixed to 10 MHz (See SP0 section).
- When **NO BOOT** mode selected, ARM7TDMI core will directly boot from external CS(0) device.
- Broadcast boot packets when the ID-code equals to 0x00.

Boot Selection via B[2:0] inputs :

B(2)	B(1)	B(0)	Mode	Config
0	0	0	NO Boot.	
1	0	0	Boot via MMU interface.	Master
1	0	1	Boot via SPI port.	Master
1	1	0	Boot via (S)UART port.	Master
0	0	1	Boot via SP0 port in chain mode.	Slave
0	1	0	Boot via SP0 port in parallel mode.	Slave

The "No Boot" is the default boot configuration, pull down resistors on the B[2:0] pins.

### 1.1.1 Multi-CDMAx boot configuration

In multi-CDMAx configuration, there are 5 possible configurations, see figure 2.

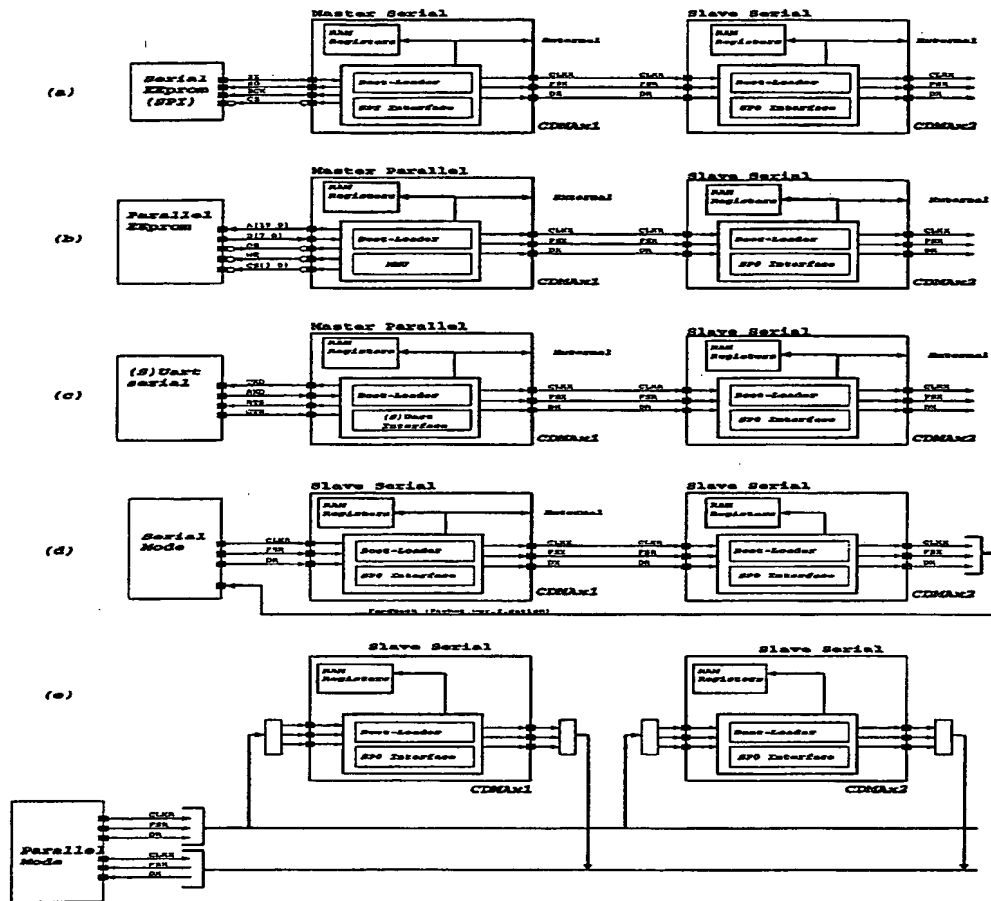


Figure 2: Multi-CDMAx Configuration

- Serial boot from SPI device and route boot data to SLAVE, see figure 2(a).
- Parallel boot from external device and route boot data to SLAVE, see figure 2(b).
- Boot from (S)UART driver and route boot data to SLAVE, see figure 2(c).
- Serial SP0 boot to SLAVE and route boot data to SLAVE, see figure 2(d).
- Serial boot from CDMA Development Accelerator board via parallel connected ASIC's, see figure 2(e).

### 1.1.2 Boot-loader configuration with SP0 link

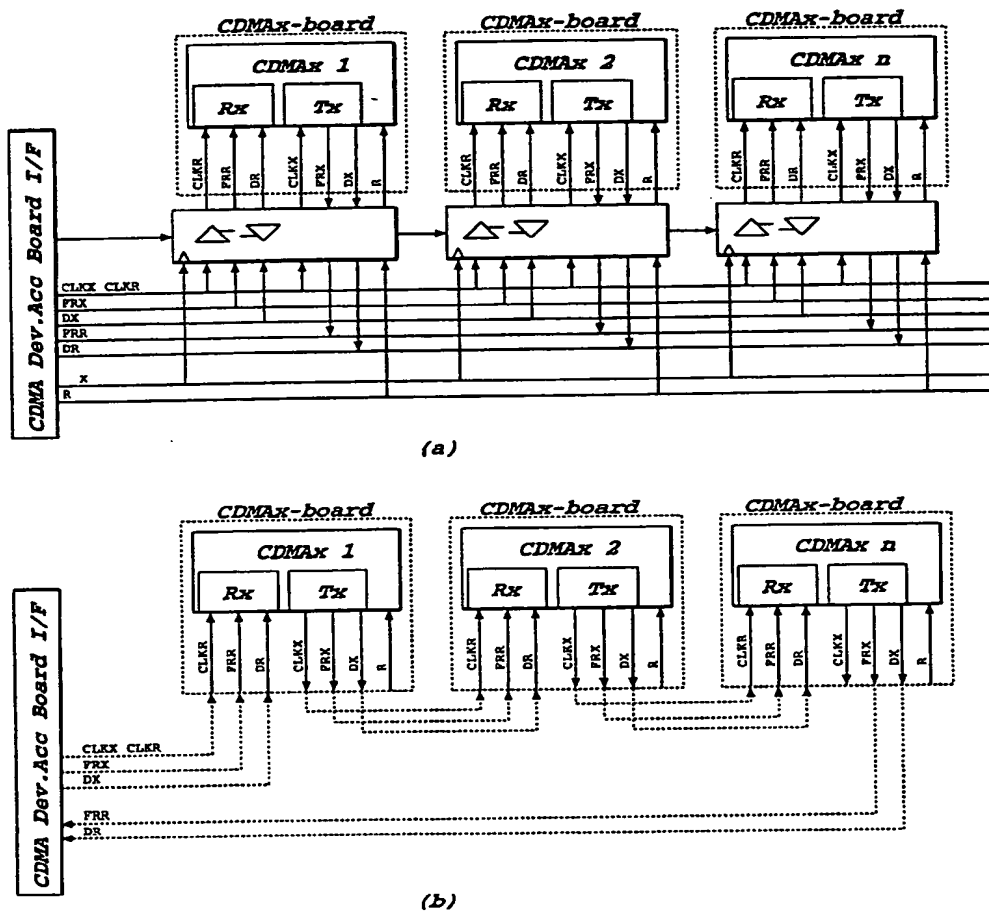


Figure 3: Boot via SP0 link

There are two boot loader configurations with the CDMA Development Accelerator board :

- Boot via parallel connected serial link channels (See figure 3(a)). The SP0 links from all chips are connected to the same common lines. The selection is done by the CDMA Development Accelerator board. Receive signals (CLKR, FRR and DR from the back plane are enabled via the *init* and *next* signals by the CDMA Development Accelerator board (See figure 4(a)). Identical for the transmit signals. Both clocks, CLKR and CLKX are inputs. When keeping the *init* signal high and insert 'x' next clock pulses, broadcast messages are possible (See figure 4(b)).

- **Boot via chain principle** (See figure 3(b)). All chips are configured in chain. When boot operation is done via a SPI or external flash device or (S)UART, the first CDMAx chip is configured in MASTER mode, the other ASIC's are configured as SLAVE. All packets except the first one, received by the MASTER are routed to the transmit side, the next SLAVE. The CLKX pin is then an output signal coming from the CLKR signal.

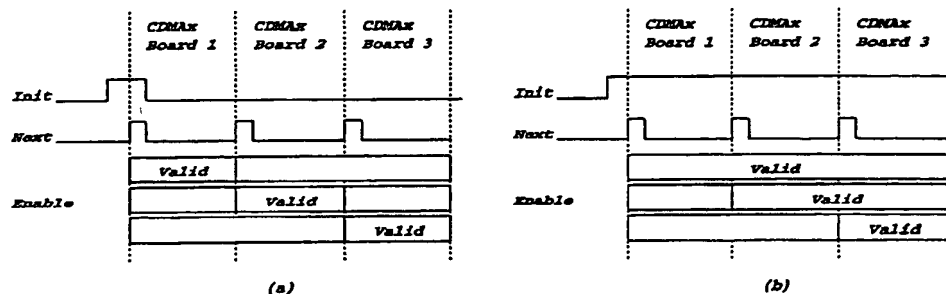


Figure 4: SP0 Parallel link principle

### 1.1.3 Boot-loader packet format

#### • Parallel channel configuration packets

The ASIC's are selected via the init and next signals as explained in the previous section. The ID bit fields are used identical as in the "Chain configuration". The format of boot data is shown in figure 5. When the length is zero, the boot operation is done. The link can be used for debug applications with the same or other packet format. Handshake mode on "request" and "acknowledge" principle is activated by software when using the link for debug applications.

#### • Chain configuration packets

There are 3 different packet modes :

- **ID-capture + boot data** : The ID of each CDMAx in the chain is NOT defined at startup. Every CDMAx ASIC will capture an ID code and boot data when receiving the first packet. This packet is NOT routed to the next CDMAx ASIC in the chain.
- **Boot data** : When all CDMAx chips in the chain are configured, selection is done via the ID code in the boot packet. When the packet ID matches the chip ID or the packet ID equals to the broadcast ID (0'h), data is captured and booted. Every packet is routed to the next CDMAx ASIC.
- **Debug data** : After booting the device, the interface can be used for debug applications. The ID-code is checked by the software running at the ARM7TDMI core. Every packet is routed to the next CDMAx ASIC. See figure 6.

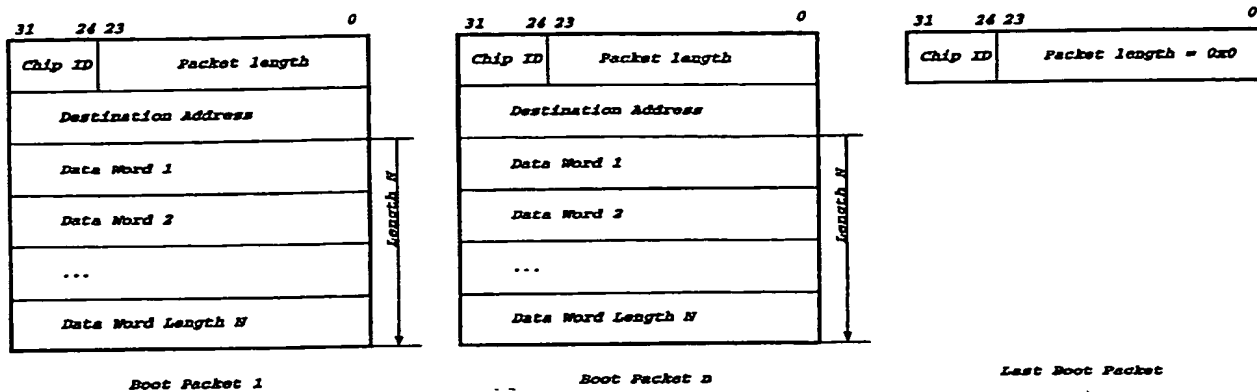


Figure 5: Boot packet format

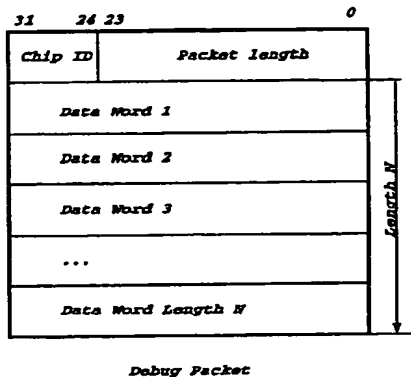


Figure 6: Debug packet format

**Remarks :**

- **Broadcast message** : When different CDMAx are in chain mode, all CDMAx will capture an ID when the first packet is received on each CDMAx. After all ID's are allocated, a broadcast message can be sent by putting the ID to 0x00.

## 1.2 SPI-Controller

The Serial Peripheral Interface (SPI) features a serial interface and software control protocol allowing operation on a simple three-wire bus. The bus signals are a clock output **SCK** plus separate data in **SI** and data out **SO** lines. Access to the SPI-compatible device is controlled through a chip select **SPLCS**. The

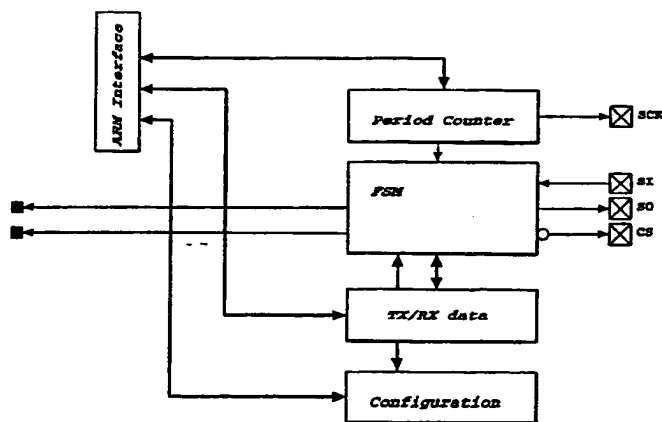


Figure 7: Serial Peripheral Interface Controller

SPI controller supports read and write operation towards a SPI compatible device. Commands and data are mixed into a single word and transmitted towards the SPI driver by the ARM7TDMI-core or boot-loader.

### Features :

- 16 address bits available for 64Kx8 SPI-device.
- Variable Clock period from 625KHz to 20MHz.
- Standalone SPI-Controller.
- Read/Write operations to SPI device.
- Page write operations with variable page width to 1024 bytes
- Configurable de-activation time of the CS signal from 1 to 16 times the half clock period.

### 1.2.1 SPI inband commands

- 1 byte write operation or page write : Configurable bytes and activation of page write via the SPI configuration register.
- 4 byte read operation (Fixed value)
- read/write status register
- Enable Write Operations
- Disable Write Operations



Command	D31-D24 : D23-D16	D15-D8	D7-D0	Note
Write data	02'h	High address byte	Low address byte	Data Byte
Read data	03'h	High address byte	Low address byte	Not Used
Write status	01'h	Status Data	Not Used	Not Used
Set write enable	06'h	Not Used	Not Used	Not Used
Reset write enable	04'h	Not Used	Not Used	Not Used
Read status	05'h	Not Used	Not Used	Not Used
				Read 1 status byte.

### 1.2.2 Events

The SPI driver will generate one dynamical event that indicates when a read operation is finished. The status of the driver is indicated by a "BUSY" bit in the configuration register, see "Register" section.

### 1.2.3 Registers

Name	Address	Reset	RW	Function
Data transmit	SPI-address + 0x0	0x00000000	RW	The transmit data register is NOT double buffered, this means that the data can be shifted.
Data receive	SPI-address + 0x4	0x00000000	R	Received data from SPI device.
Configuration	SPI-address + 0x8	0x00063C00	RW	Control register. Default 625kHz period and 25.6 us de-activation time.

#### • Data transmit register

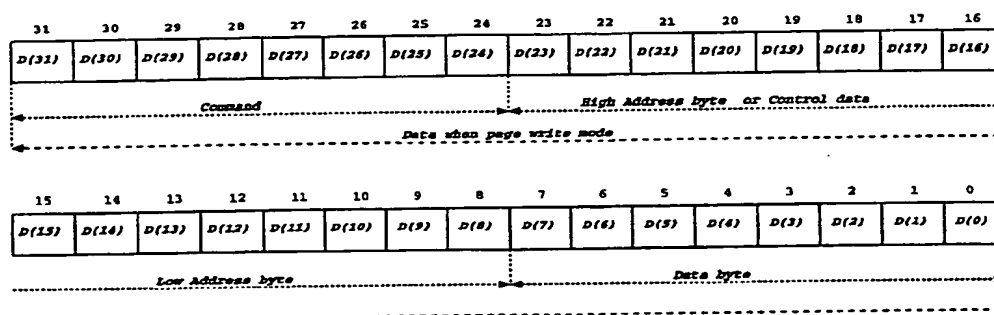


Figure 8: Data transmit register

- Data receive register

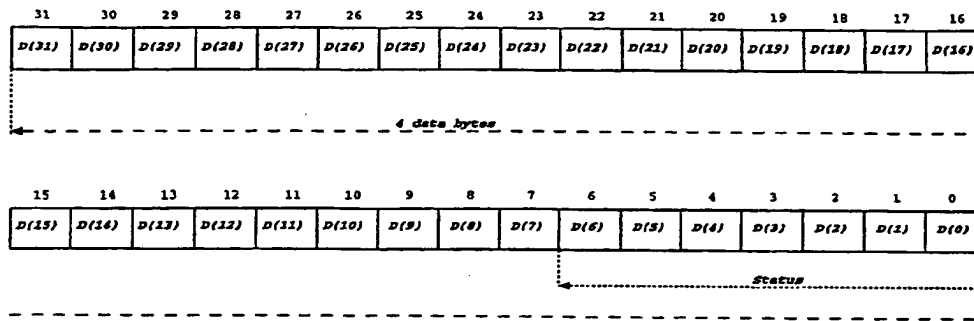


Figure 9: Data receive register

- Configuration register

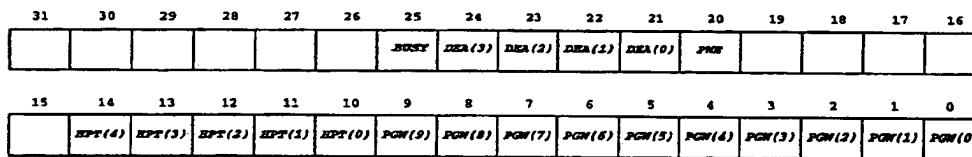


Figure 10: Configuration register

Bit	Name	Reset	Function
9-0	PGW	0-0	Page width in bytes, from 1 to 1024 bytes.
14-10	HPT	0X1F	Half Period Time value, half period = HPT * 25 ns (40MHz MCLK)
20	PWE	0	'1' = page write enable.
24-21	DEA	0XF	De-activation time, CS high timing = half period time * DEA
25	BUSY	0	READ only, indicates when SPI state machine is running.
31-26	RESERVED	0-0	Write '0'

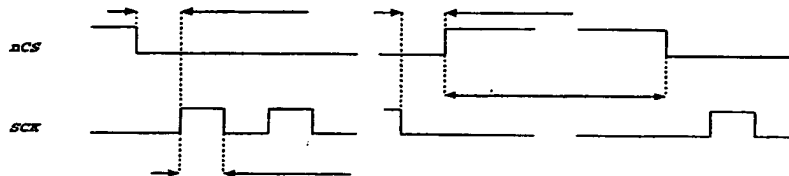


Figure 11: CS and SCK configurable timings

## 1.2.4 Timing

Figure 12(a) shows the MSB and LSB locations of the incoming/outgoing bit stream. The write and read operations are shown in figure 12(a) and figure 12(b).

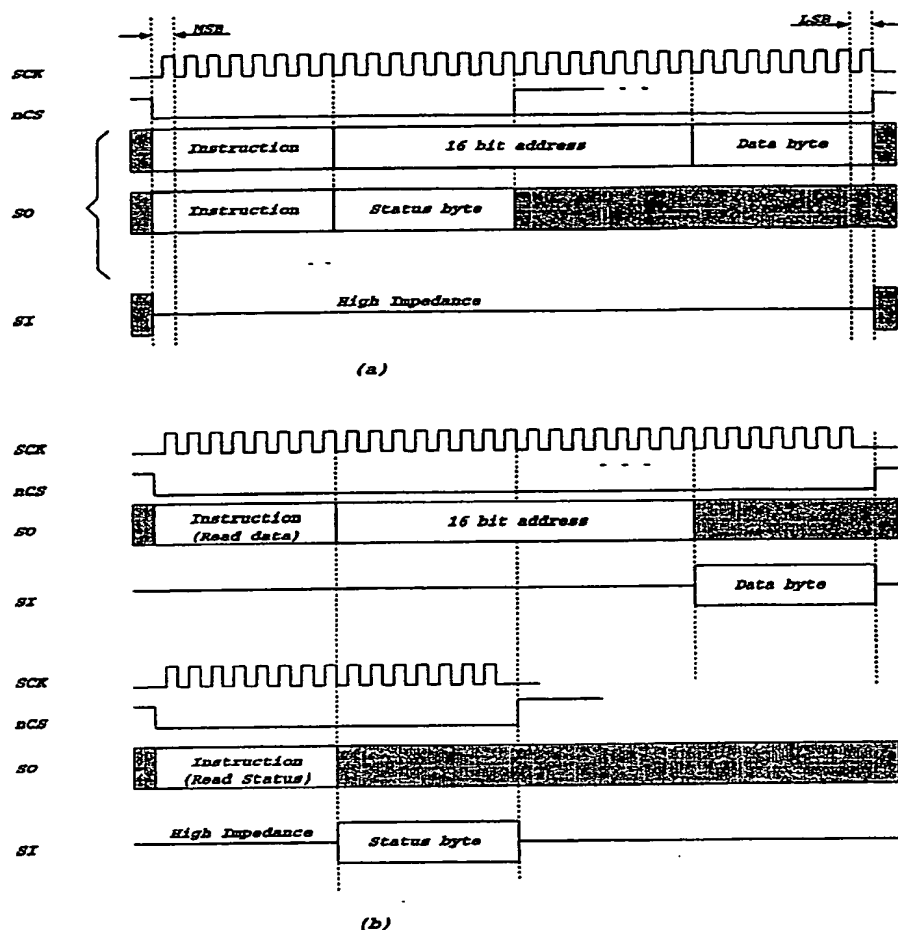


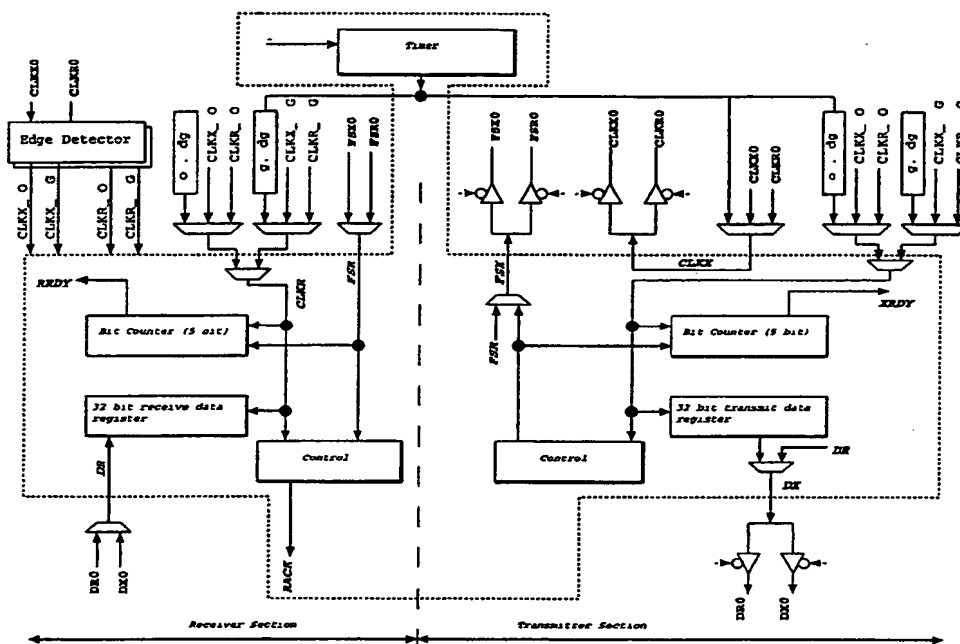
Figure 12: SPI timing

## Remarks:

- *Write Enable* : The external SPI device will power up in write disable state. All programming instructions must therefore be preceded by a *Write Enable* instruction.
- *HOLD* : A external *HOLD* pin is foreseen by several SPI devices. It allows the master to pause the communication from the slave device. Not available in the CDMAx.

The Serial Port operates as a bidirectional synchronous data link that interfaces with the internal bus. The main features are :

- Compatible with the TMS320C\* DSP devices in the fixed burst mode operation.
- Flowcontrol with "ACK" and "REQUEST" in-band control.
- 32 bit data transfer simultaneously in both directions.
- Clock source from internal, external.
- Flexible clock rate from 600 Hz to 10 MHz.
- Receive and transmit event generation.
- Direction control of all signals when Multi-CDMAx chain-configuration selectable.



**Figure 13: *SP0* block diagram**

The SP0 port contains 6 pins for bidirectional data transfer, 3 receive and 3 transmit bus signals (see figure 13). The internal receiver/transmitter signals are :

- **FSR0** : The input frame synchronization pulse . This pulse indicates the start of the receive data process over DR.
- **CLKR0** : Serial Port 0 receive input clock . Serial shift clock of incoming data.
- **DR0** : Data receive input signal .
- **FSX0** : The frame synchronization input/output pulse . This pulse indicates the start of the transmit data process over DX.

- CLKX0 : Serial Port 0 transmit clock . Serial shift clock of outgoing data.
- DX0 : Data transmit signal .

The internal FSR signal is the result of the selection between FSX0 or FSR0 signals as shown in figure 13. Identical for all other internal receiver/transmitter signals.

The incoming clock signals CLKR and CLKX are synchronized with the internal system clock.

When the serial port is used in Multi-CDMAx slave mode, the receive signals are routed to the transmit signals or vica versa. In the Multi-CDMAx master mode, the CLKX0 is configured as output with a default period of 4 MHz. When the SP0 port is used as a debug interface, the clock period is configurable.

### 1.3.1 Events

The SP0 creates 2 dynamical events, receive and transmit event. Both events are maskable and are an OR function of several events.

- Receiver :
  - RRDY : Receive 32 bit boot DATA.
  - RACK : Receive ACK packet.
- Transmitter :
  - XRDY : Transmit 32 bit boot DATA, last bit shifted out.

### 1.3.2 SP0 registers

Name	Address	Reset	RW	Function
Transmit Data	SP0-address + 0x0	0x00000000	RW	Transmitted data.
Receive Data	SP0-address + 0x4	0x00000000	R	Received data.
Timer	SP0-address + 0x8	0x00000000	RW	CLKx period
Control - status	SP0-address + 0xC	0x000001F8	RW	Control and event values

#### • Transmit Data Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
D(31)	D(30)	D(29)	D(28)	D(27)	D(26)	D(25)	D(24)	D(23)	D(22)	D(21)	D(20)	D(19)	D(18)	D(17)	D(16)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D(15)	D(14)	D(13)	D(12)	D(11)	D(10)	D(9)	D(8)	D(7)	D(6)	D(5)	D(4)	D(3)	D(2)	D(1)	D(0)

Figure 14: SP0 transmit data register

Bit	Name	Reset	Function
31-0	D	0	Transmit data register.

#### • Receive Data Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
D(31)	D(30)	D(29)	D(28)	D(27)	D(26)	D(25)	D(24)	D(23)	D(22)	D(21)	D(20)	D(19)	D(18)	D(17)	D(16)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D(15)	D(14)	D(13)	D(12)	D(11)	D(10)	D(9)	D(8)	D(7)	D(6)	D(5)	D(4)	D(3)	D(2)	D(1)	D(0)

Figure 15: SP0 receive data register

Bit	Name	Reset	Function
31-0	D	0	Receive data register. Read only.

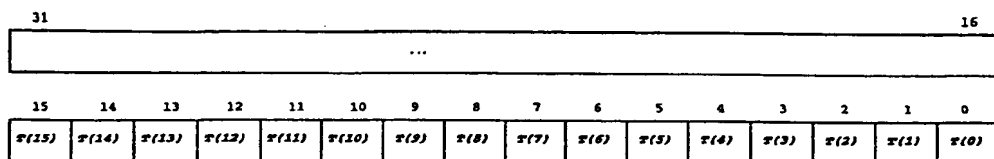


Figure 16: SP0 timer register

- **Timer register**

Bit	Name	Reset	Function
15-0	T	0xA	CLKx periods from 600 Hz to 20 MHz. Default 4 MHz.
31-16	RESERVED	0-0	Read as 0.

- **Control-Status register**

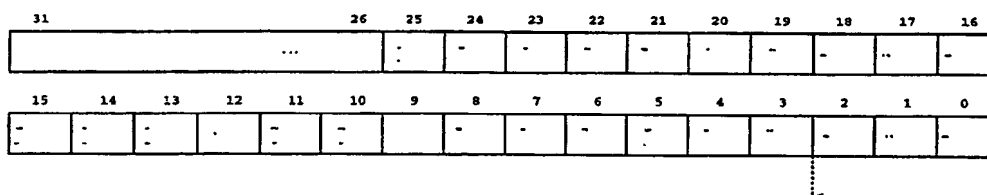


Figure 17: SP0 control-status register

Bit	Name	Reset	Function
0	RRDY	0	Read only. The receiver has new data available.
1	RACK	0	Read only. The receiver has received a "ACK" bit.
2	XRDY	0	Read only. XRDY='1' when last bit of the 32 bit data is shifted out or after reset procedure. XRDY value change to '0' when data is written to the data register. This bit will NOT indicate when an "ACK" is received during handshake mode. The transmitter will wait until an "ACK" is received from the other device.
3	FSXOUT	1	This bit configures the FSX0 pin as an input (FSXOUT='1') or an output (FSXOUT='0'). Read after boot will indicate the direction of FSX0. In Master mode always output when booting.
4	DXOUT	1	This bit configures the DX0 pin as an input (DXOUT='1') or an output (DXOUT='0'). Read after boot will indicate the direction of DX0. In Master mode always output when booting.
5	CLKXOUT	1	This bit configures the CLKX0 pin as an input (CLKXOUT='1') or an output (CLKXOUT='0'). Read after boot will indicate the direction of CLKX0. In Master mode always output when booting.
6	FSROUT	1	This bit configures the FSR0 pin as an input (FSROUT='1') or an output (FSROUT='0'). Read after boot will indicate the direction of FSR.
7	DROUT	1	This bit configures the DR0 pin as an input (DROUT='1') or an output (DROUT='0'). Read after boot will indicate the direction of DR0.
8	CLKROUT	1	This bit configures the CLKR0 pin as an input (CLKROUT='1') or an output (CLKROUT='0'). Read after boot will indicate the direction of CLKR0.
9	HS	0	If HS='1', the handshake mode is enabled. If HS='0', the handshake mode is disabled. No handshaking when booting.
10	XCLKSRCE	0	If XCLKSRCE='1', the internal transmit clock is used. If XCLKSRCE='0', the external transmit clock is used via pin CLKX0.
11	RCLKSRCE	0	If RCLKSRCE='1', the internal receive clock is used. If RCLKSRCE='0', the external receive clock is used via pin CLKR0.
12	TACK	0	Transmit "ACK" message. Bit is automatically cleared by hardware.
13	DXROUT	0	If DXROUT='0', DR0 signal routed to DX0. If DXROUT='1', data coming from internal data register. Valid after boot operation.
14	FXROUT	0	If FXROUT='0', FSR0 signal routed to FSX0. If FXROUT='1', synchronization coming from internal. Valid after boot operation.
15	CLKXROUT	0	If CLKXROUT='0', CLKR0 signal routed to CLKX0. If CLKXROUT='1', clock coming from internal timer.
16	MRRDY	0	Mask on RRDY signal, mask = '0' active
17	MRACK	0	Mask on RACK signal, mask = '0' active
18	MXRDY	0	Mask on XRDY signal, mask = '0' active
19	ETXACK	0	Enable ACK message transmit when the receive buffer is full and read by the software.
20	CLEAR	0	Clear the receive and data path of the SP0 and all internal registers. Active '1', software controlled when clearing is finished.
21	BURSTCLK	0	If '1', outgoing clock is in burst mode. Clock active when frame synchronization and data valid.
22	POLCLKR	0	If '1', invert the polarity of CLKR0.
23	POLCLKX	0	If '1', invert the polarity of CLKX0.
31-24	RESERVED	0-0	Read as 0.

## 1.3.3 Handshake Mode

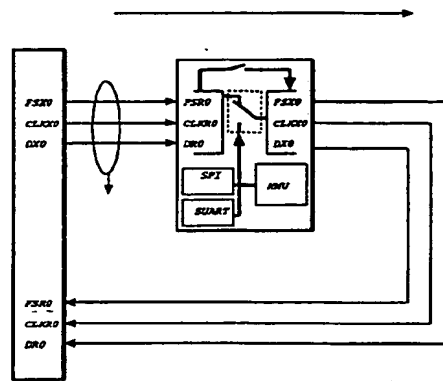


Figure 18: SP0 handshake principle

The receive bus signals of the CDMAx are connected to the transmit signals of the debug/boot tool. When the device is in boot operation, handshaking mechanism is NOT enabled. During the debug operation, the debug/boot tool will send a data packet with a leading '1' (Request) to the CDMAx. He will receive this packet and confirmed it with sending a single '0' to the debug/boot tool, and vica versa. Dataflow control is done by the "ACK" operations on both sides. The transmission of "ACK" has a higher priority then transmitting a data packet. See figure 19.

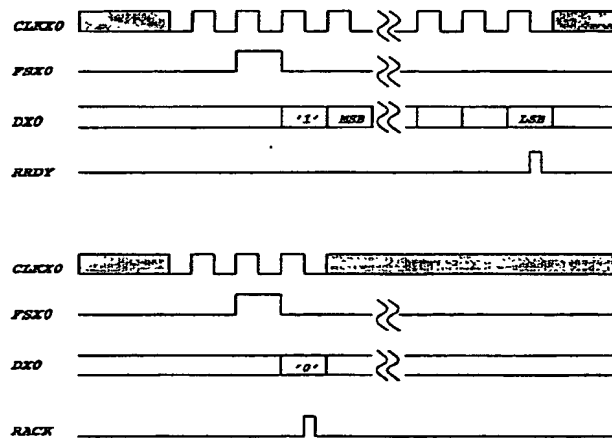


Figure 19: Ack-Request handshake principle



## 1.3.4 Multi-CDMAx mode

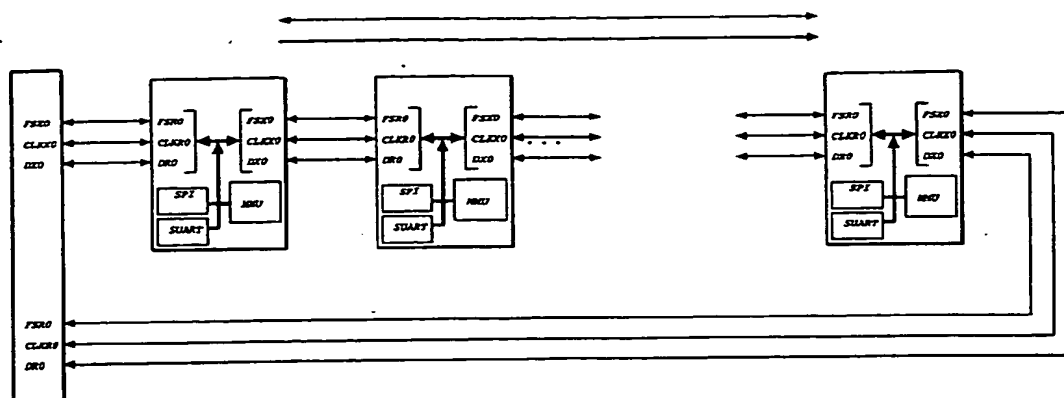


Figure 20: SP0 multi-CDMAx boot directions

## 1. Boot application:

- **Slave mode :** When the boot selection bits B[2:0] are configured to 0x1 (Boot via SP0 chain method), the signals CLKR0 and CLKX0 are sampled during boot mode. All SP0 port signals are input after reset. When an edge is detected on one of the CLK signals. This signal is configured as input, the other CLK signals defined as output.
- **Master mode :** The CLKX0 is configured as an output clock with a clock period of default 4 MHz.

Data is routed through the SP0 driver in boot mode after the ID-capture packet is received. When "x" CDMAx devices are in chain configuration, "x" ID packets are sent through the chain until a CDMAx will capture the ID packet. Each CDMAx will capture a unique ID value. The next packets are routed through the full chain and can be verified by the user.

B[2:0]	Master Slave	Signals	Remark
0x4 or 0x5 or 0x6	Master	FSX0, CLKX0, DX0 : outputs FSR0, CLKR0, DR0 : inputs	CLKX0 internal clock
0x1	Slave	Sample CLKX0 and CLKR0	Direction signals defined via sample method. The transmit clock equal to the receive clock.
0x2	Slave	FSX0, DX0 : outputs FSR0, CLKR0, CLKX0, DR0 : inputs	

Figure 21 shows the different ID packets during boot and the "ripple through" principle when using a multi-CDMAx configuration. The reference configuration of figure 20 is used as example.

**Remark:**

All SP0 signals FSx are external connected with a pull-down resistor. The FSR is an active high signal and is in the "SP0 chain boot (0x1)" an input during startup.

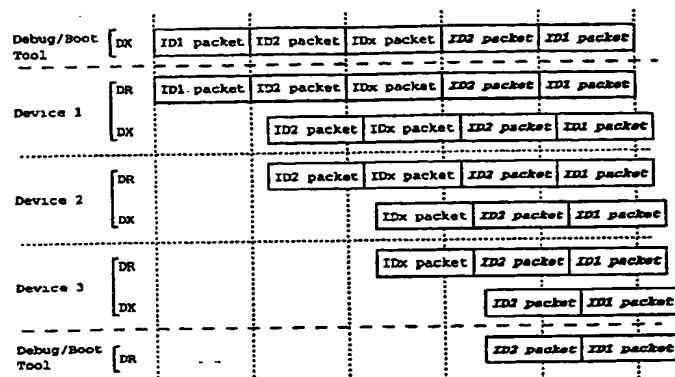


Figure 21: SP0 boot example

## 2. Debug application:

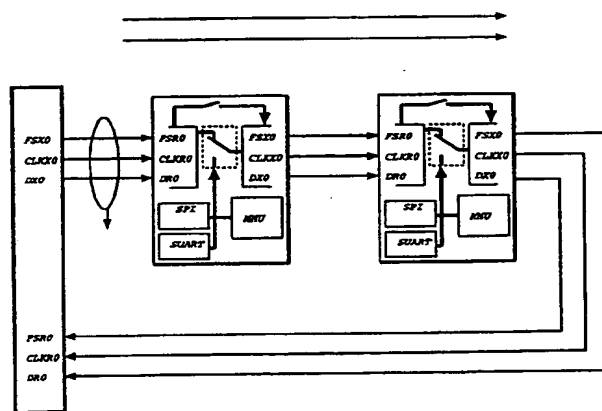
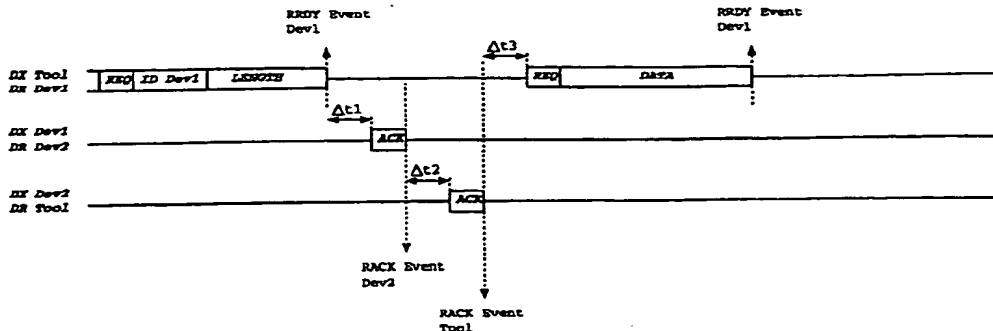
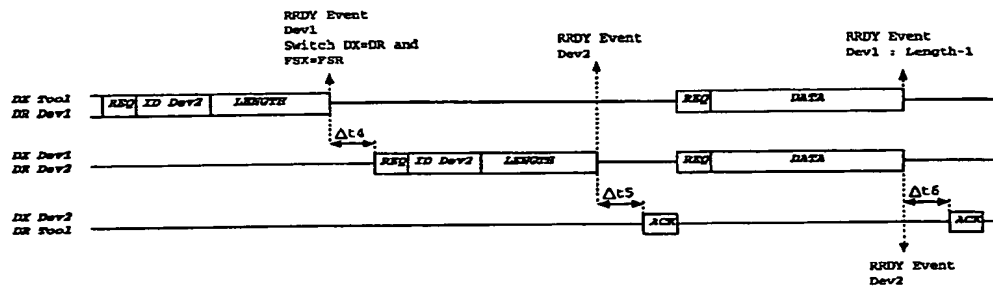


Figure 22: SP0 multi-CDMAx debug chain configuration

Figure 22 shows a multi-CDMAx configuration with boot mode 0x4, 0x5, 0x6 or 0x1 used as debugging link after the boot procedure. The direction of all SP0 signals is defined in the boot procedure and will be preserved during the debug procedure. The next steps are necessary to startup a debug access when more then one CDMAx is in chain configuration. There are 2 modes:

- Device is selected (ID code packet = ID code device). See figure 23.
  - (a) Default settings : DX internal, CLKX internal, FSX internal. Set by DXROUT and FXROUT to '1' and CLKXROUT = '1'.
  - (b) Set the ETXACK bit to '0', NO automatically ACK transmit after reading out the receive data register.
  - (c) Wait until RRDY event activated.
  - (d) First 32 bit received contains the ID en length of the packet. Always capture length of packet. ID = device ID.
  - (e) Set TACK bit to '1'. ACK message will be transmitted.
  - (f) Wait until RRDY event activated.
  - (g) 32 bit data received, decrement length and check if equals to 0 and goto (e).

Figure 23: *SP0 debug example A*Figure 24: *SP0 debug example B*

- Device is NOT selected (ID code packet  $\neq$  ID code device). See figure 24.
  - (a) Default settings : DX internal, CLKX internal, FSX internal. Set by DXROUT and FXROUT to '1' and CLKXROUT = '1'.
  - (b) Set the ETXACK bit to '0', NO automatically ACK transmit after reading out the receive data register.
  - (c) Wait until RRDY event activated.
  - (d) First 32 bit received contains the ID en length of the packet. Always capture length of packet. ID  $\neq$  device ID. Write 32 bit to data register. 32 bits are transmitted to the next device.
  - (e) Wait until XRDY event activated.
  - (f) Switch to DX=DR, FSX=FSR and CLKX=CLKR. Set by DXROUT and FXROUT to '0' and CLKXROUT = '0'.
  - (g) Wait until RRDY event activated.
  - (h) 32 bit data received, decrement length and check if equals to 0. If 0 then to the default settings of DX, CLKX and FSX by clearing the DXROUT, FXROUT and CLKXROUT bits. If  $\neq$  0 then goto (g).
  - (i) If ACK received, this event should be masked.

**Remark:**

See figure 23 and figure 24. The timing  $\Delta t1$  gives the process timing of the debug application when receiving the first word of a packet. The received data is read, ID code checked and length saved. If ID is equal, then activate the TACK bit. When ID does not match, then the word is written to the data register and re-transmitted to the next device,  $\Delta t$  between receive RRDY event first device and RRDY event next device is approx.  $64 \text{ bit baudrate timing} + \Delta t4$ . When the second word is received by a device that is NOT selected, the data is routed from the FR to the FX pin, without extra delay. The device will receive a RRDY event

*and will decrement the length counter and switch the FX, DX and CLKX signals back to the default values. When large packets are transmitted, the packet is only delayed during the first word.*

662220" 92454705

# SECRET

- IO's can be enabled as an output or input.
- An optional input glitch filtering is available.
- Each signal can be programmed to generate an interrupt when a level change occurs.

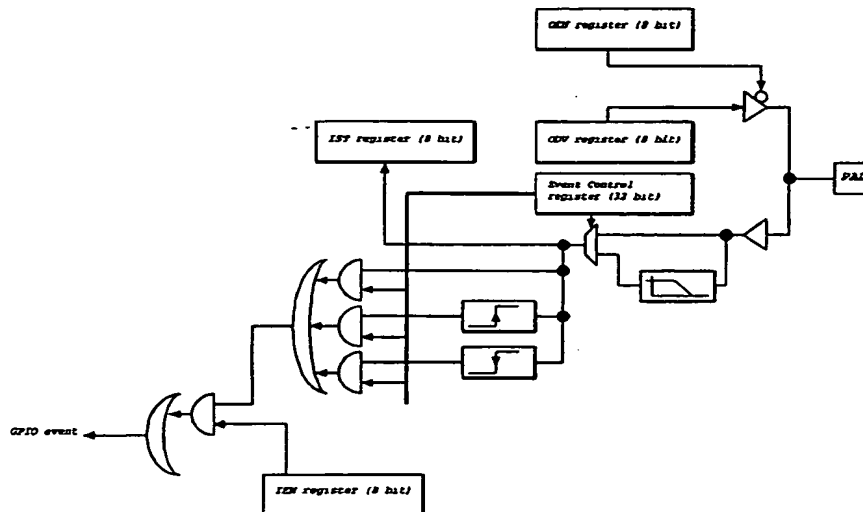


Figure 25: *GPIO blockdiagram*

Figure 25 shows the GPIO block functionality per line. Each individual I/O is associated with a bit position in the GPIO user interface registers. Each of these registers are 8 bits wide. Each line can be enabled by the OEN bit in the 8-bit OEN-register. The drive value is controlled by the ODV bit in the ODV-register. The pad status is defined in the IST-register. The GPIO event is an or function of all maskable lines. The input event generation can be selected to positive, negative or the input signal.

Name	Address	Reset	RW	Function
Data Control	GPIO-address + 0x0	0x000000FF	RW	Data configuration settings.
Event Control	GPIO-address + 0x4	0x00000000	RW	Event configuration settings.

- *Data GPIO control register*

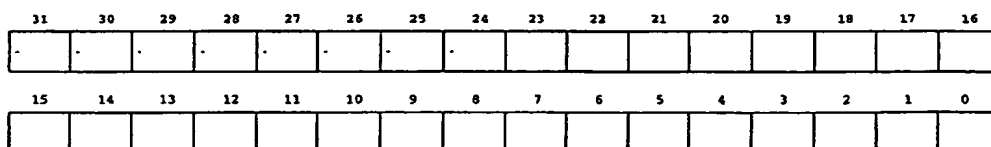


Figure 26: *Data GPIO control register*

Bit	Name	Reset	Function
7-0	OEN	0xFF	Output ENable bit fields. '0' is output enabled, '1' is output disabled.
15-8	IEM	0x00	Input Event Mask bit fields. '0' is mask, '1' is un-mask. The 8 inputs are masked with the associated mask bit and ored to one general GPIO event.
23-16	IST		Input SStatus bit fields. Reflects the value on the GPIO pads.
31-24	ODV	0x00	Output Data Valid bit fields. Drive value on the GPIO pad when the associated bit in the OEN bit field is enabled.

• **Event GPIO Control register**

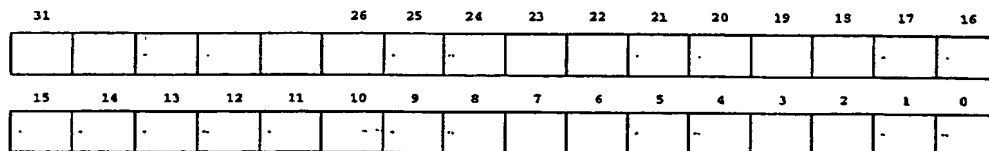


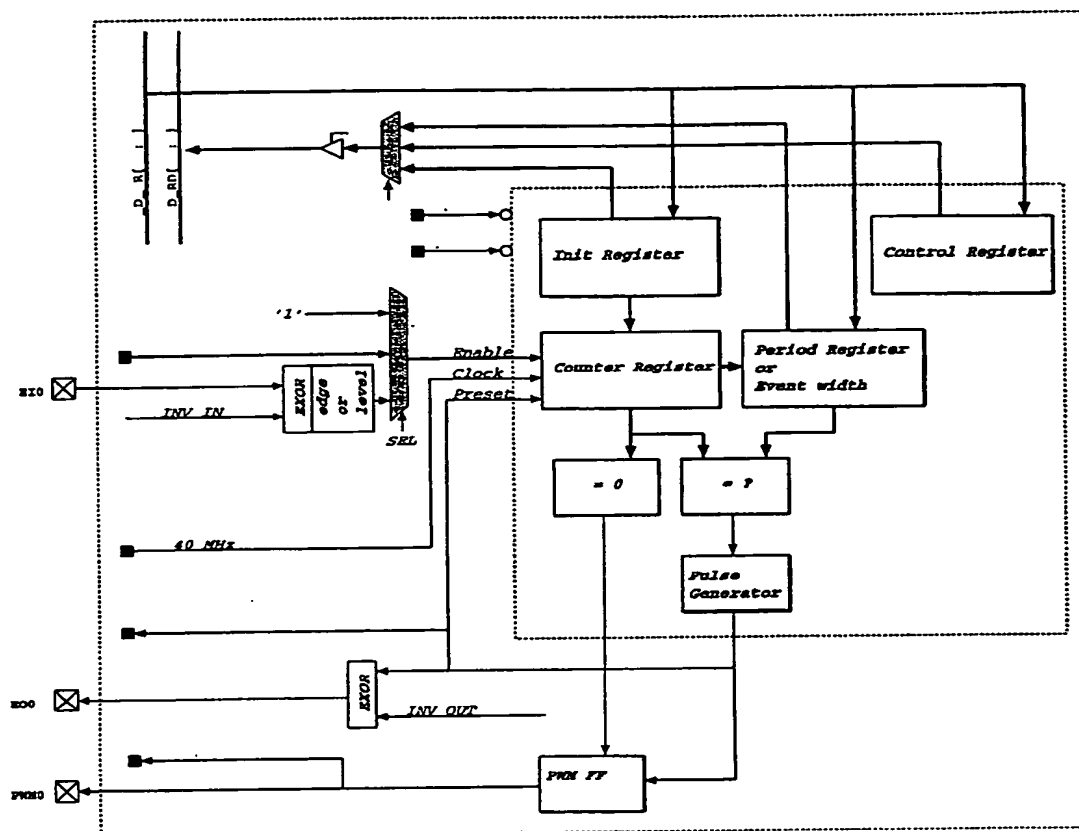
Figure 27: Event GPIO control register

Bit	Name	Reset	Function
0	FIL	0	Filter select. No filtering on input signal when '0', else filtering with a latency of 4 master clock periods.
1	NOR	0	Route input signal to the event generator.
2	POS	0	Route positive edge result to the event generator.
3	NEG	0	Route negative edge result to the event generator.
31-4			All other bits are associated on the GPIO signals and having the same function as bit field [3:0].

**Remark:**

See figure 25. When the input of a GPIO signal is routed thru the GPIO cell without reclocking (NOR selected and un-masked), the GPIO input should be synchronized with the system clock and one period active.

SECRET



**Figure 28: PWM-Timer blockdiagram**

The PWM-Timer module are general-purpose 32-bit timer/event counters, with two modes and internal or external clocking. The PWM-Timer module can be used as a internal signal generator or by the external world.

The main features are:

- Timer applications of 32 bits.
- PWM puls generation with 8 bit accuracy.
- Width measurement applications.

The PWM-Timer module has the following external signals :

- **EIO** : External Input event. This input is internal invertable and is synchronized with the master clock.
- **EOO** : External Ouput event. This event is generated and invertable.
- **PWM0** : PWM signal.

The PWM-Timer can be configured in 3 modes :

- **PWM mode** : The module generate a PWM puls with a duty cycle between 0 and 100 %. The 8-bit counter is clocked with the internal 40 MHz clock.
- **Timer Mode** : The timer has a configurable period value. Dynamical event generation when compare equal of counter registers with period register and "auto" preset of counter register.
- **Event Width measurement** : The counter register will increment each time the enable signal is active. The input events or the "External Event IN" or the "Internal Event IN". The value in the counter register will be copied to the period register when the input event is zero.

The clock enable source is selectable in both modes. See table .

CLS(1)	CLS(0)	Source	Remark
0	0	Always '1'	Default
0	1	Internal event	
1	0	External event	

The external event source can be inverted and is synchronized. This means that 3 40MHz clock periods are needed for synchronization the external event input as showed in figure 29.

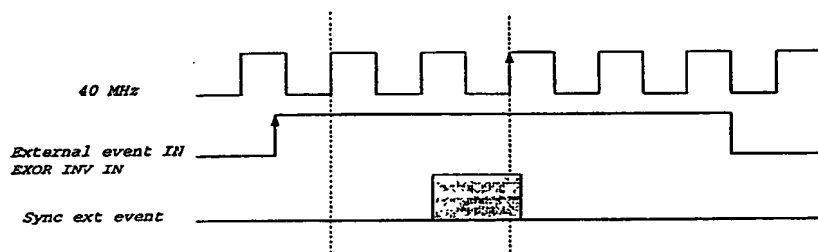


Figure 29: PWM-Timer synchronization

Two dynamical events are generated :

- **Internal Event Out** : Event generated in the Timer mode.
- **Internal PWM Out** : Event generated in the PWM mode.

### 1.5.1 PWM-Timer registers

Name	Address	Reset	RW	Function
Control	(S)PWMTimer-address + 0x0	0x00000000	RW	Settings.
Period	(S)PWMTimer-address + 0x4	0x00000000	RW	Period register value
Init	(S)PWMTimer-address + 0x8	0x00000000	RW	Init register value

- **Control Register**

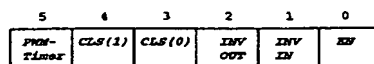


Figure 30: Control register



Bit	Name	Reset	Function
0	EN	0	'1' = start the PWM-Timer module.
1	INV IN	0	'0' = input normal, '1' = input inverted.
2	INV OUT	0	'0' = output normal, '1' = output inverted.
4-3	CLS	00	CLock enable Source. See table.
5	PWM-Timer	0	'0' = Timer, '1' = PWM.
6	Width-meas	0	'0' = Off, '1' = On.
31-7	RESERVED	0-0	Read as 0.

• *Period Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PW(31)	PW(30)	PW(29)	PW(28)	PW(27)	PW(26)	PW(25)	PW(24)	PW(23)	PW(22)	PW(21)	PW(20)	PW(19)	PW(18)	PW(17)	PW(16)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PW(15)	PW(14)	PW(13)	PW(12)	PW(11)	PW(10)	PW(9)	PW(8)	PW(7)	PW(6)	PW(5)	PW(4)	PW(3)	PW(2)	PW(1)	PW(0)

Figure 31: Period register

Bit	Name	Reset	Function
31-0	PW	0x0	The Period register also contains the width measurement value.

• *Init Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SV(31)	SV(30)	SV(29)	SV(28)	SV(27)	SV(26)	SV(25)	SV(24)	SV(23)	SV(22)	SV(21)	SV(20)	SV(19)	SV(18)	SV(17)	SV(16)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SV(15)	SV(14)	SV(13)	SV(12)	SV(11)	SV(10)	SV(9)	SV(8)	SV(7)	SV(6)	SV(5)	SV(4)	SV(3)	SV(2)	SV(1)	SV(0)

Figure 32: Init register

Bit	Name	Reset	Function
31-0	SV	0x0	A 32 bit register that contains the start count value of the Counter register.

## 1.6 (S)UART driver

The CDMAx provides a full-duplex, synchronous/asynchronous receiver/transmitter that interfaces with the internal bus. The main features are:

- Programmable Baud Rate Generator. Baud rates from 2K4 to 460K8.
- Receive and Transmit Event generation on word or byte basis.
- 8-bit character length with 1 start bit and 1 or 2 stop bits.
- 8 Byte FIFO at receive data path.
- 32-bit buffer at transmit path.
- Break detection.
- Break transmission under software control.
- Dataflow control, RTS-CTS or Xon-Xoff.
- Auto Bauding of Baud Rates between 4K8 and 115K2.

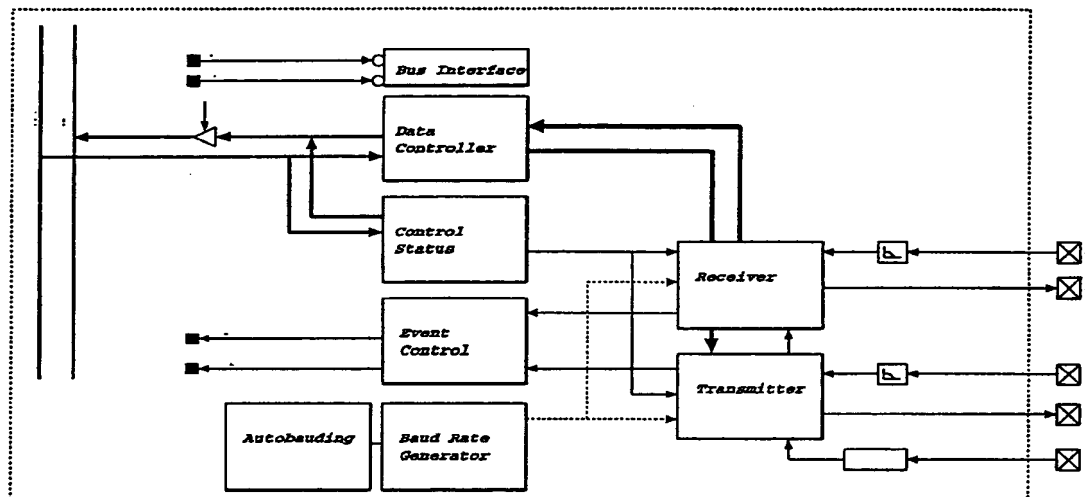


Figure 33: (S)UART blockdiagram

The (S)UART has the following external signals:

- RXD : Receive Serial Data is an input.
- TXD : Transmit Serial Data is an output.
- SCLK : The Serial clock is an input.
- CTS : Active low "Clear To Send" input indicates driver is ready to exchange data.
- RTS : Active low "Request To Send" output indicates driver is ready to exchange data.

The Baud Rate Generator provides the bit period clock named the Baud Rate clock to both the Receiver and the Transmitter. The Baud Rate is selected by the Baud Rate Register and Baud Rate source clock or detected by the autobaud module.

BRCL(1)	BRCL(0)	Baud Rate Clock	Note
0	0	MCLK	Default
0	1	MCLK/4	
1	0	MCLK/16	
1	1	MCLK/64	

### 1.6.1 Flow-control

- Hardware : By CTS and RTS signals.
- Software : By "Xon" and "Xoff" in-band commands.

DFC(1)	DFC(0)	Flow-control	Note
0	0	NO	Default
0	1	CTS-RTS	
1	0	Xon-Xoff	Configurable value

The receive threshold is fixed at 6 bytes. The receiver automatically executes a dataflow mechanism when the threshold value is reached. The mechanism is selected by the DFC bit fields.

### 1.6.2 (S)Uart events

The Uart driver will generate two dynamical events, receive and transmit event. Both events contain an OR function of several maskable bits. See figure 34.

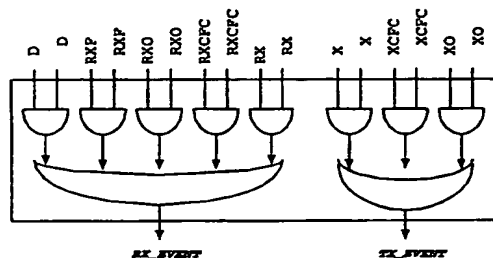


Figure 34: Event Mask principle

#### • Receiver

Name	Function
RE	Byte or word received. Figure 35(a) shows the exact timing when the receive event is generated. Depends on the stopbits value in the configuration register.
RXCFC	Change flowcontrol bits. When a "Xon" or "Xoff" message is received, the RXCFC will be set. When the "CTS" signal is de-activated, RXCFC is set. Status of both dataflow mechanism is indicated by the status register.
RXO	Overflow receiver. The receiver contains a 8 byte FIFO mechanism. When 6 bytes are stored into the FIFO, the receiver will automatically send "Xoff" byte or de-activate the "RTS" signal when dataflow is enabled. At the same time an overflow event is generated. When reading out the receive buffer, the "Xon" message will be transmitted or "RTS" will be activated.
RXFE	Framing error. The receiver will generate a framing error when a stopbit is received with a '0' value. The databits are always stored in the receiver buffer.
BD	Break Detection. The break condition is detected by the receiver when all data, and stop bits are low. At the moment of the low stop bit detection, the receiver asserts the Break Detection event bit. End of receive break is detected by a high level of incoming signal. Event generated at start of break.

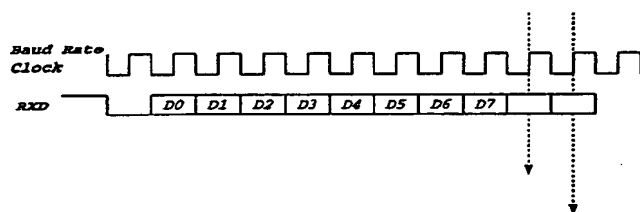
- Transmitter

Name	Function
TE	Byte or word transmitted. The event is generated during the last bit shift operation. Figure 35(b) shows the exact timing. When one stopbit is selected, the update time between event generation and sending the next byte without added stopbits is maximum 2 Baud Rate periods ( $\Delta t_1$ ). When 2 stopbits are configured, the maximum update time will be 3 Baud Rate periods ( $\Delta t_2$ ).
TXCFC	Change flowcontrol bits. When a "Xon" or "Xoff" message is received, the TXCFC will be set. When the "RTS" signal is de-activated, TXCFC is set. Status of both dataflow mechanism is indicated by the status register.
TXO	Overflow transmitter. The transmitter contains a 4 byte register. When 4 bytes are stored into the register and new data is written to this register before all bytes are transmitted, the transmitter overflow event is generated. Will be cleared by reading out the event register. The transmitted word can be corrupted by the seconde write operation.

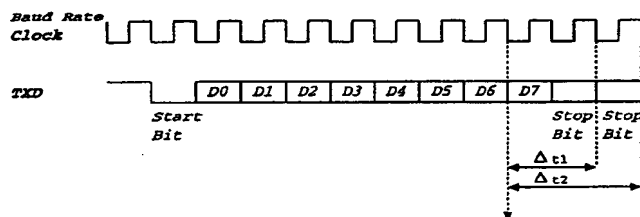
There are general status bits for the receiver and transmitter.

- Byte Receive Counter. Indicates the amount of received bytes in the FIFO.
- CTS and RTS signal levels.
- Xon-Xoff indicator for receive and transmit path.

See "(S)UART registers" section for more detailed information about the event bits and the mask values.



(a)



(b)

Figure 35: Event timings

### 1.6.3 Asynchronous Timing

The Receiver continuously waits for a valid start bit. When one has been detected, the receiver samples the RXD at the theoretical mid-point of each bit, see figure 36(a). The transmitter transmits start bit, data bits and stop bits serially, least significant bit first on the falling edge of the serial clock, see figure 36(b).

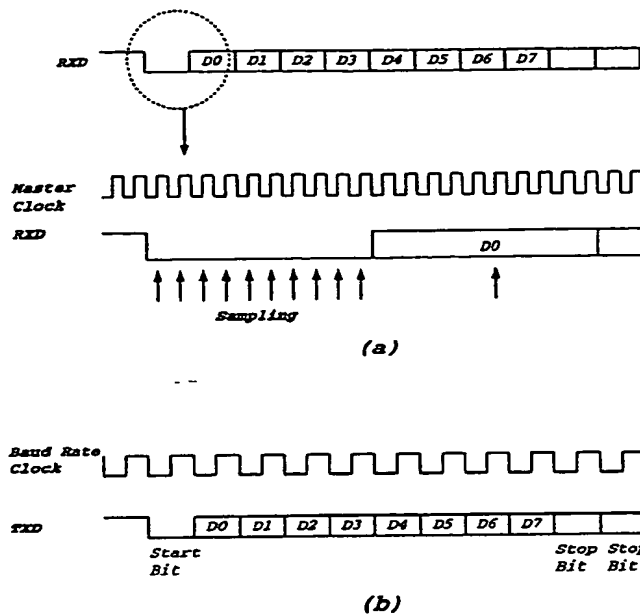


Figure 36: Asynchronous timings

#### 1.6.4 Synchronous Timing

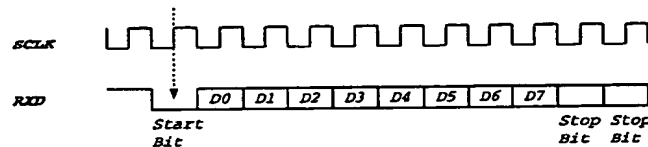


Figure 37: Synchronous timings

The Receiver samples the RXD signal on each rising edge of the Baud Rate Clock. If a low level is detected, it is considered as a start bit. Data bits and stop bits are sampled and the receiver waits for the next start bit. The Transmitter operates the same as in the asynchronous mode, data is transmitted at the falling edge of SCLK. The SCLK is synchronized with the master clock. This means that 2 master clock periods are added before sampling of the RXD data. Same principle for the transmitter side. The SCLK signal can be inverted when the SCLK-INV bit in the Control register is activated.

### 1.6.5 Auto Bauding

The autobaud module samples the RXD signal with a sample clock of 625 KHz ( If the MCLK equals to 40MHz). The autobaud character will be the "Carriage Return" value (0D'h) and is validated by the module. Baud Rates between 4K8 and 115K2 are detected. The sample counter of the startbit during autobauding is readable via the status register. Software control is possible by reading out this value and overwrite the Baud Rate Register value. The BRCL bit field is configured to "10" when autobauding is enabled.

666220-92451709

## 1.6.6 (S)UART registers

Name	Address	Reset	RW	Function
Control	(S)UART-address + 0x0	0x00000188	RW	Settings.
Mask	(S)UART-address + 0x4	0x00000000	RW	Mask events
Transmit Data	(S)UART-address + 0x8	0x00000000	RW	Data to transmit
Receive Data	(S)UART-address + 0xC	0x00000000	R	Received data
Baud Rate	(S)UART-address + 0x10	0x00000104	RW	Baud Rate settings
Event and status	(S)UART-address + 0x14	0x00000000	R	Event values

## • Control register

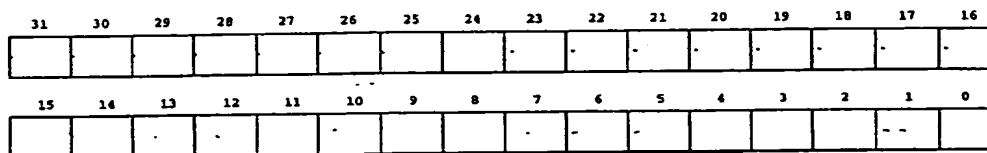


Figure 38: Control register

Bit	Name	Reset	Function
1-0	BRCL	00	Baud-Rate CLock source bits.
2	Stop bits	0	'0' = 1 Stop bit, '1' = 2 Stop bits.
3	Sync-Async	1	'0' = Synchronous mode (Use of SCLK), '1' = Asynchronous mode (Use of Baud Rate Generator).
4	Clear	0	Clear all internal (S)UART flags and counters. Active during rising edge of clear bit.
6-5	DFC	00	Data Flow-Control. See table.
7	RE	1	'0' = Receiver byte event, '1' = word event (32 bits).
8	TE	1	'0' = Transmitter byte event, '1' = word event.
9	XBR	0	Transmit break. Continuously break transmission when XBR is active.
10	SCLK-INV	0	Invert the SCLK input signal.
11	AB	1	Activate autobauding.
12	ABCL	0	Clear autobauding settings and restart by software when AB is active.
13	ABAC	0	Enable to clear autobauding settings and restart by hardware when receiving a break.
14	TXXON	0	When '1', it will transmit the Xon value, auto-cleared.
15	RESERVED	0	Write '0'.
23-16	Xon	0-0	Xon value. (Example Ctrl-O = 0x11)
31-24	Xoff	0-0	Xoff value. (Example Ctrl-S = 0x13)

## • Mask register

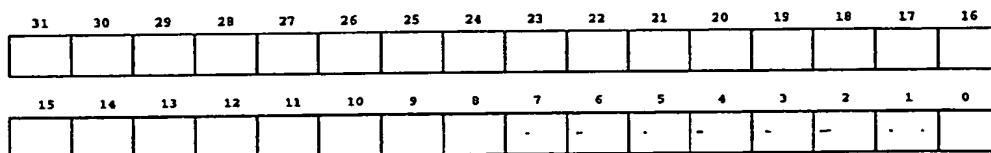


Figure 39: Mask register

Bit	Name	Reset	Function
0	MBD	0	Mask Break Detection.
1	MRXE	0	Mask Receiver Event byte or word.
2	MRXCFC	0	Mask Receiver Change of Flow-Control.
3	MRXO	0	Mask Receiver Overflow .
4	MRXFE	0	Mask Receiver Framing Error .
5	MTXE	0	Mask Transmitter Event byte or word .
6	MTXCFC	0	Mask Transmitter Change of Flow-Control .
7	MTXO	0	Mask Transmitter Overflow .
31-8	RESERVED	0-0	Write '0'.

• *Transmit Data register*

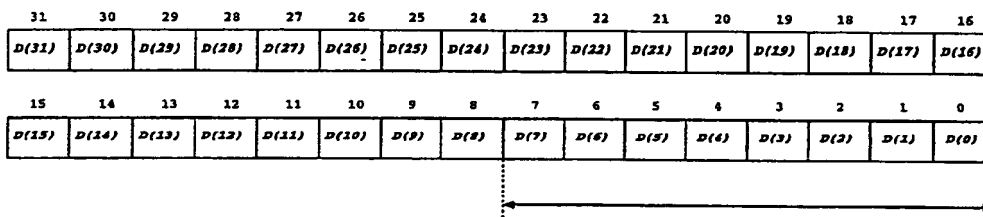


Figure 40: *Transmit Data register*

Bit	Name	Reset	Function
31-0	D	0	Transmit data values.

• *Receive Data register*

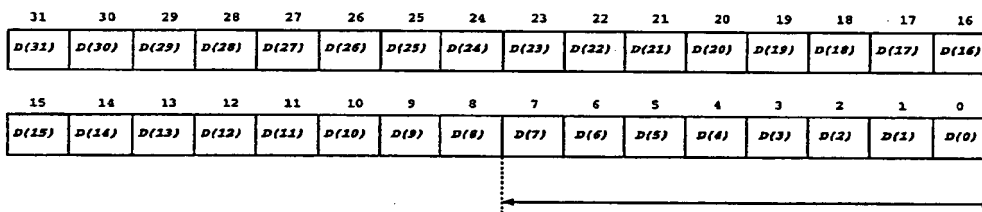


Figure 41: *Receive Data register*

Bit	Name	Reset	Function
31-0	D	0	Received data.



- Baud Rate register

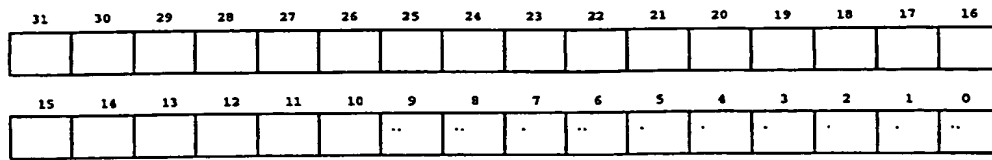


Figure 42: Baud Rate register

Baud Rate clock = MCLK (40Mhz)					Baud Rate clock = MCLK/4 (10Mhz)				
Baud Rate(K)	KBAUD	%Error	BRVAL ('h)	Note	Baud Rate(K)	KBAUD	%Error	BRVAL ('h)	Note
2.4	NA				NA				
4.8	NA				NA				
9.6	NA				NA				
14.4	NA				14.4	14.404	0.06 %	0x02B6	
19.2	NA				19.2	19.230	0.16 %	0x0208	
38.4	NA				38.4	38.461	0.16 %	0x0104	
57.6	57.636	0.06 %	0x02B6		57.6	57.803	0.35 %	0x00AD	
115.2	115.273	0.06 %	0x015B		115.2	116.279	0.94 %	0x0056	
230.4	231.213	0.35 %	0x00AD		230.4	232.558	0.94 %	0x002B	
460.8	465.116	0.94 %	0x0056		460.8	476.190	3.34 %	0x0015	

Baud Rate clock = MCLK/16 (2.5Mhz)					Baud Rate clock = MCLK/64 (0.625Mhz)				
Baud Rate(K)	KBAUD	%Error	BRVAL ('h)	Note	Baud Rate(K)	KBAUD	%Error	BRVAL ('h)	Note
NA					2.4	2.4038	0.16 %	0x0104	
4.8	4.8077	0.16 %	0x0208	DEFAULT	4.8	4.8077	0.16 %	0x0082	
9.6	9.6154	0.16 %	0x0104		9.6	9.6154	0.16 %	0x0041	
14.4	14.450	0.35 %	0x00AD		14.4	14.534	0.94 %	0x002B	
19.2	19.230	0.16 %	0x0082		19.2	18.939	-1.36 %	0x0021	
38.4	38.461	0.16 %	0x0041		38.4	39.062	1.73 %	0x0010	
57.6	58.139	0.94 %	0x002B		57.6	56.818	-1.36 %	0x000B	
115.2	119.047	3.34 %	0x0015		115.2	125.000	8.51 %	0x0005	
230.4	250.000	8.51 %	0x000A		230.4	208.333	-9.58 %	0x0003	
460.8	500.000	8.51 %	0x0005		460.8	625.000	35.63 %	0x0001	

- Event and status Register

The receiver and transmitter event bits are cleared when reading out the Event-Status register.

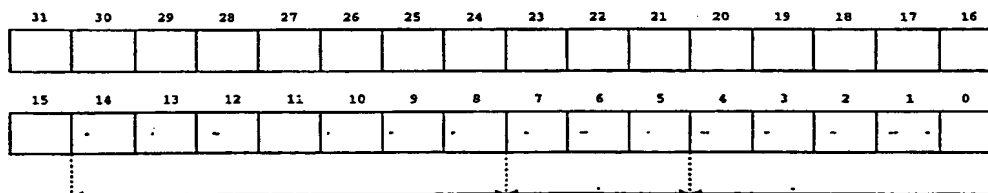


Figure 43: Status register

Bit	Name	Reset	Function
0	RXE	0	Receive event (byte or word event).
1	RXCFC	0	Change of Flow-Control on receive side.
2	RXO	0	Receiver overflow.
3	RXFE	0	Receiver Framing Error.
4	RXBD	0	Receiver Break Detection.
5	TXE	0	Transmitted event (byte or word event).
6	TXCFC	0	Change of Flow-Control on transmit side.
7	TXO	0	Transmitter overflow.
10-8	BRC	0-0	Byte Receive Counter. Amount of bytes in FIFO.
11	CTS	0	Clear To Send signal level.
12	RXXC	0	Xon ('0') - Xoff ('1') status receiver, '1' after power-up.
13	RTS	0	Request To Send signal level.
14	TXXC	0	Xon ('0') - Xoff ('1') Control transmitter, '1' after power-up.
15	ABST	0	Autobaud status, autobaud done when '1'.
25-16	ABCT	0-0	Autobaud counter value.
31-26	RESERVED	0-0	Read as 0.

**Remarks: Byte and Word transmit and receive and RXD sampling**

- **Transmit :** A 32 bit words is written to the data register when a single byte should be transmitted. Bit 0 to 7 contains the byte data. The (S)UART will transmit the data and generate a event during the stopbit (if TE bit in the control register is '0'). The (S)UART will wait until data is written to the data register, meanwhile a stopbit value is transmitted.
- **Receiver :** When the RE bit in the control register is '0', an event is generated each time a byte is received. When the FIFO of 8 bytes is filled with 6 bytes and the dataflow control is enabled, the RTS is de-activated or a Xoff byte is automatically transmitted. RTS is toggled or Xon is transmitted when reading the data register.
- **RXD sampling :** The RXD signal is sampled with an internal Baud Rate clock that depends on the BRCL bits and the master clock frequency. The maximum sample fault is the fault made by dividing the master clock to the internal Baud Rate clock and a fixed added fault of maximum one internal Baud Rate clock + 2 master clock periods. Example : BRCL = "10" and Baud Rate 9600. The fault is equal to  $1/9600$  divided by  $clk/16 + clk/16 + 2$  master clock periods.

Baud Rate	Samples	Range	Hex value
4K8	520	[570-470]	[0x23A-0x1D6]
9K6	260	[285-235]	[0x11D-0x0EB]
14K4	173	[190-156]	[0x0BE-0x09C]
19K2	130	[143-117]	[0x08F-0x075]
38K4	65	[71-59]	[0x047-0x03B]
57K6	43	[50-38]	[0x032-0x026]
115K2	21	; 38	; 0x026

- **Auto Bauding sample :**



The main features are :

- 4 Chip Selects with a range of 16 M .
- Byte, half word and word operations to internal/external devices.
- Upper and lower byte enable when byte operations to a 16 bit device.
- External/Internal ZERO wait states access, 25 ns clock period. External device, Taa maximum 10 ns.
- Wait state generator from 0 ns to 375 ns.
- External "WAIT" signal from slow devices. Invert by "INV nWAIT" signal in configuration register.
- "WAIT" combination of external and/or wait state generator signal.
- Memory remap when NOBOOT signal active.
- External address tri-state control, power reduction.
- Automatic waitstate generation when consecutive accesses are made to two different external devices.
- ABORT generation to the ARM7TDMI when access to an address outside any memory page.

### 1.7.1 External signal list

The external MMU signals are :

Name	Description	Type	Name	Description	Type
A0-A23	Address bus	O	nUBE	Upper Byte Enable	O
D0-D15	Data bus	I/O	nOE	Output Enable	O
nWE	Write Enable	O	nCS0-nCS3	Chip Selects	O
nLBE	Lower Byte Enable	O	nEWAIT	External WAIT	I

### 1.7.2 Internal signal list

The signals between the ARM7TDMI core and the MMU are :

Name	Type	Description	Function
wait	O	NOT wait	Wait generation towards the ARM7TDMI. When the core is accessing slow devices, it can be made to wait for an integer number of MCLK cycles by driving "wait" LOW.
mas(1:0)	I	Memory Access Size	Indicates when a word transfer or a half-word or byte length is required. Valid for both read and write operations. The signals take the value 10'b for words, 01'b for half words and 00'b for bytes.
rd	I	NOT read-write	HIGH value indicates a write cycle, LOW a read cycle of the ARM7TDMI.
bl(3:0)	O	Byte Latch Control	Indicates when data and instructions are latched from the external data bus. bl(3) indicates that D(31:24) is valid, identical for bl(2) to bl(0).
mreq	I	NOT memory request	A LOW indicates that the ARM7TDMI requires memory access during the following cycle.
abort	O	Memory Abort	Valid when the requested output is NOT allowed.

### 1.7.3 External Memory Mapping

The MMU provides 4 chip lines. See figure 45.

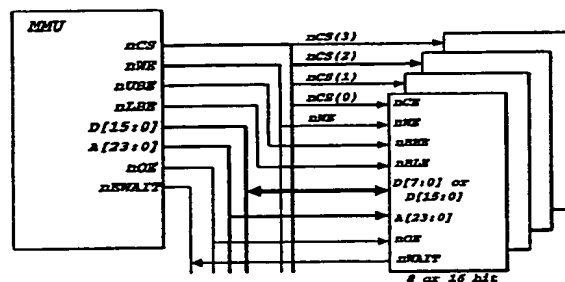


Figure 45: Memory connections for 4 external devices

#### • Wait configuration

The MMU can automatically insert wait states. The different type of wait states are listed below and are depending on the Wait State Selection (WSS) bits :

WSS(1)	WSS(0)	Wait state depends on	Info	Remark
0	0	nEWAIT		
0	1	WSV		
1	0	nEWAIT AND WSV	AND function of both inputs	The slowest of both
1	1	nEWAIT OR WSV	OR function of both inputs	The fastest of both

Default value : nEWAIT AND WSV

Remarks :

- When using the nWAIT AND WSV wait state generation, the minimum value of WSV should be 2. Identical When the nWAIT OR WSV is selected, WSV should be greater or equal then 1. The nWAIT signal will be used when slow external devices are connected to the CDMAx.

- The *nWAIT* signals is NOT synchronized internally with the MCLK. This means that the timing of the *nWAIT* should apply to the MMU timings. *nWAIT* is checked with the falling edge of the MCLK.

- **Selectable Bus Width**

Data bus width of 8 or 16 bits can be selected for each chip select line. These settings are configurable into the DBW-bit of the MMU register.

DBW	Data Bus Width
0	8-bit data bus width
1	16-bit data bus width

662240" 92454709

- Selectable wait states

The Wait States Value (WSV) of each chip select is controlled by the WSV fields in the MMU register. The wait state generator depends on 2 inputs, nEWAIT and the internal Wait State Counter (WSC). The wait states range can be selected from 0 to 15 times the master clock period.

WSV	Time	WSV	Time	WSV	Time	WSV	Time
0x0	0 ns	0x4	100 ns	0x8	200 ns	0xC	300 ns
0x1	25 ns	0x5	125 ns	0x9	225 ns	0xD	325 ns
0x2	50 ns	0x6	150 ns	0xA	250 ns	0xE	350 ns
0x3	75 ns	0x7	175 ns	0xB	275 ns	0xF	375 ns

- Swap memory map

The SWAp bit in the MMU control register configures the external chip selects CS(0) and CS(1) in the global memory map. When '1', CS(1) is located between 0x00000000 and 0x0FFFFFFF and CS(0) between 0xD0000000 and 0xDFFFFFFF. Caution when the NOBOOT signal is '0'. See "Memory Map" section.

## 1.7.4 MMU Registers

- CS configuration register

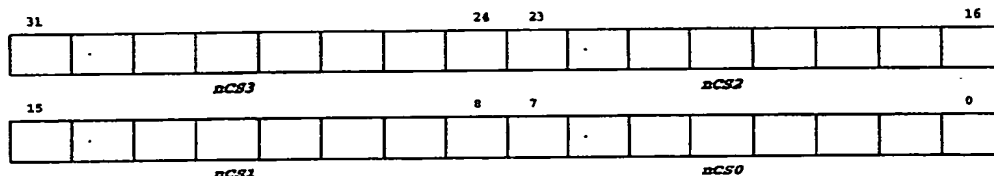


Figure 46: MMU-CSS Register

Bit	Name	Reset	Function
3-0	WSV	0xF	CS(0) wait state value.
5-4	WSS	0x2	CS(0) wait state selection bit field.
6	DBW	0x0	CS(0) databus width.
7	INV nEWAIT	0x0	CS(0) Invert nEWAIT signal, '1' invert active.
3-0	WSV	0xF	CS(1) wait state value.
5-4	WSS	0x2	CS(1) wait state selection bit field.
6	DBW	0x0	CS(1) databus width.
7	INV nEWAIT	0x0	CS(1) Invert nEWAIT signal, '1' invert active.
3-0	WSV	0xF	CS(2) wait state value.
5-4	WSS	0x2	CS(2) wait state selection bit field.
6	DBW	0x0	CS(2) databus width.
7	INV nEWAIT	0x0	CS(2) Invert nEWAIT signal, '1' invert active.
3-0	WSV	0xF	CS(3) wait state value.
5-4	WSS	0x2	CS(3) wait state selection bit field.
6	DBW	0x0	CS(3) databus width.
7	INV nEWAIT	0x0	CS(3) Invert nEWAIT signal, '1' invert active.

- General configuration register

The MMU will automatically generate wait states when consecutive accesses are made to two different external devices or switching from read from read to a write operation to the same device, see figure 48 and figure 49. The amount of wait states is selected by the WSR bit field in the MMU-GC register. The default value is 3'd. Individual WSR bit fields are foreseen per external chip select(TBC).

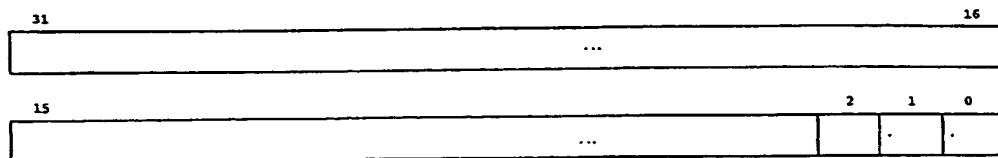
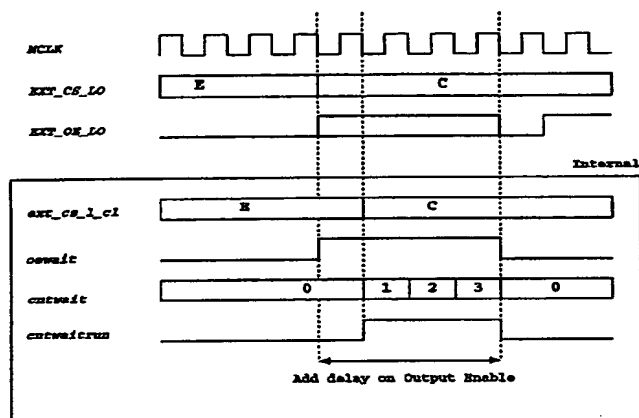


Figure 47: MMU-GC Register

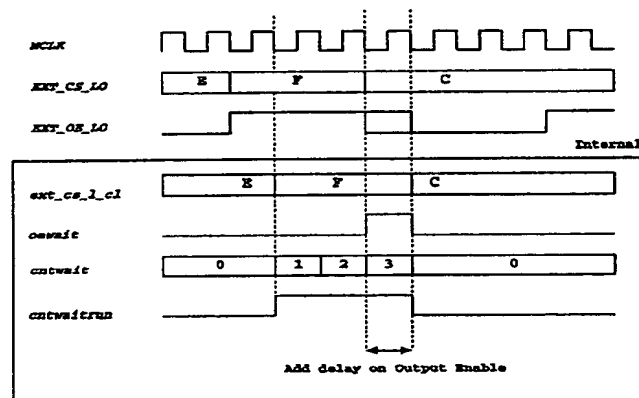
Bit	Name	Reset	Function
1-0	WSR	0x3	Wait states when consecutive accesses to two different external devices.
2	SWA	0x0	Swap CS(1) to CS(0) location when NOBOOT signal active.

### 1.7.5 External Timing

- Wait states between two consecutive external read operations. See figure 48.



(a)



(b)

Figure 48: Output Enable wait timing

Figure 48(a) shows consecutive read operation two different devices. The default value of 3 wait states are added to the external output enable signal. Figure 48(b) shows consecutive operations with a



dummy cycle between. The wait state insertion value is reduced to 2 clock periods when the default value of 3'd in the WSR field is used.

**Remark:**

*When consecutive write to read operation is done to the same external device. Assume that the delay of the tri-state buffers on the data pins are faster ( from '1' or '0' to 'Z' value) then the access time of the external device. NO wait states added.*

66220-92454T09

- Wait states between two consecutive external read/write operation. See figure 49.

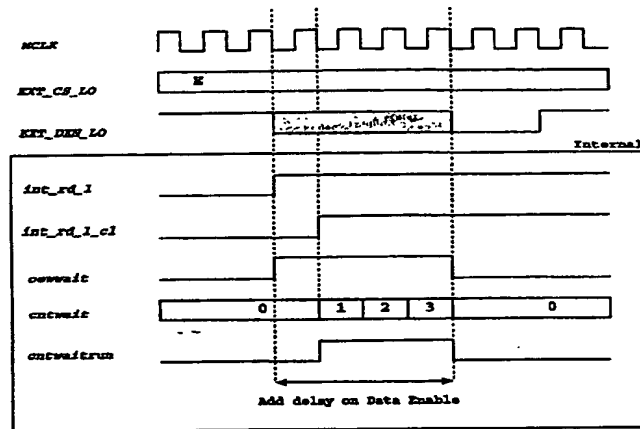


Figure 49: Data Enable wait timing

66220-925109

- 0, 1, x wait states timing. See figure 50 a, b and c.

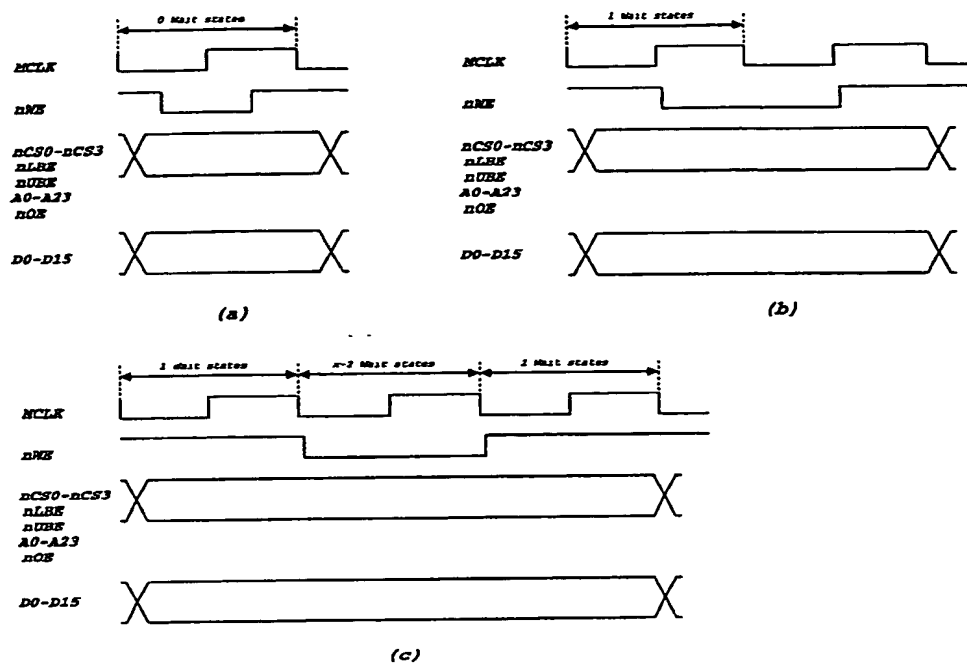


Figure 50: External MMU timing for x wait states access

- External WAIT timing. See figure 51.

When the WSS value is configured where the "nEWAIT" signal is included, 2 clock periods are added before the external WAIT is internally verified. Figure 51 shows the timing of the "check" and "verify" clock periods of the MMU.

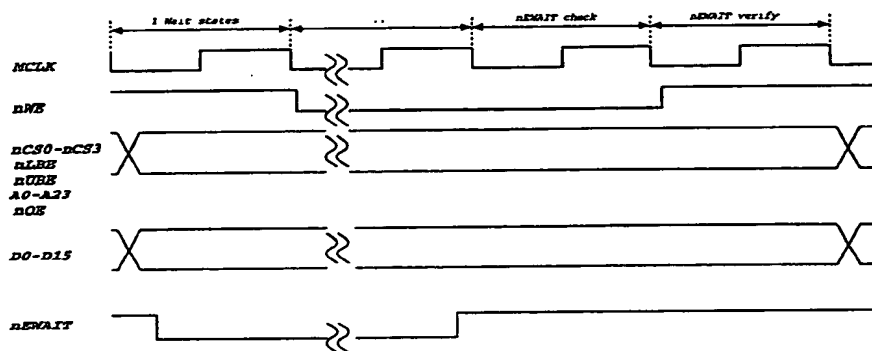


Figure 51: External MMU timing with nEWAIT

## 1.8 ARM-Interface

The ARM7TDMI is a general purpose 32-bit microprocessor, which offer high performance for very low power consumption. The core runs at a maximum frequency of 40 MHz.

The main features are :

- 40 MHz maximum frequency.
- ARM or THUMB mode.
- Debug by the ICEBreaker module via JTAG interface.
- Debug software by including the "ANGEL" debug C-code of ARM into the user software.

### 1.8.1 Internal interface

The ARM7TDMI used connections are:

662240-92HMT09

Name	Type	Description	Function
MCLK	I	Memory clock input	Connected to the master clock PLL circuit output. Maximum frequency of 40 MHz.
nWAIT	I	Not wait	MMU will activate the nWAIT signal when needed, depends on the memory access and the defined wait states.
nIRQ	I	Not interrupt request	Connected to the Interrupt Controller unit. Synchronous interface.
nFIQ	I	Not fast interrupt request	Idem.
ISYNC	I	Synchronous interrupts	Always '1'.
nRESET	I	Not reset	Controlled by the Boot-Loader unit. Active after power-up and de-activated when boot operations is done.
BUSEN	I	Data bus configuration	Always '1'. Select the unidirectional data busses.
BIGEND	I	Big endian configuration	Always '0'. Little Endian memory format.
nENIN	I	Not enable input	Always '0'. Data bus always enabled.
ABE	I	Address bus enable	Always '1'. Address bus always enabled.
APE	I	Address pipeline enable	Always '0'. De-pipelined mode selected. <i>Note: When (S)DRAM interface is implemented, this signal can be '1'.</i>
ALE	I	Address latch enable	Connected to the INVERT MCLK.
DBE	I	Data Bus Enable	Always '1'.
TBE	I	Test Bus Enable	Always '1'.
DBGREQ	I	Debug request	Dedicated Input Pin. (See ARM7TDMI Data Sheet.)
BREAKPT	I	Breakpoint	Dedicated Input Pin. (See ARM7TDMI Data Sheet.)
DBGACK	O	Debug acknowledge	Dedicated Output Pin. (See ARM7TDMI Data Sheet.)
nEXEC	O	Not executed	Dedicated Output Pin. (See ARM7TDMI Data Sheet.)
EXTERN1	I	External input 1	Dedicated Input Pin. (See ARM7TDMI Data Sheet.)
EXTERN0	I	External input 0	Dedicated Input Pin. (See ARM7TDMI Data Sheet.)
DBGEN	I	Debug enable	Dedicated Input Pin. (See ARM7TDMI Data Sheet.)
RANGEOUT0	O	ICEbreaker Rangeout0	Dedicated Output Pin. (See ARM7TDMI Data Sheet.)
RANGEOUT1	O	ICEbreaker Rangeout1	Dedicated Output Pin. (See ARM7TDMI Data Sheet.)
DBGREQI	O	Internal debug request	Dedicated Output Pin. (See ARM7TDMI Data Sheet.)
COMMRX	O	Communications Channel Receive	Dedicated Output Pin. (See ARM7TDMI Data Sheet.)
COMMTX	O	Communications Channel Transmit	Dedicated Output Pin. (See ARM7TDMI Data Sheet.)
TCK	I	Test clock	Dedicated Input Pin. JTAG interface.
TMS	I	Test Mode Select	Dedicated Input Pin. JTAG interface.
TDI	I	Test Data Input	Dedicated Input Pin. JTAG interface.
nTRST	I	Not Test Reset	Dedicated Input Pin. JTAG interface.
TDO	O	Test Data Output	Dedicated Output Pin. JTAG interface.
DRIVEBS	O	Boundary scan cell enable	Boundary scan interface.
ECAPCLKBS	O	Exttest capture clock	Boundary scan interface.
ICAPCLKBS	O	Intest capture clock	Boundary scan interface.
nHIGHZ	O	Not HIGHZ	Boundary scan interface.
PCLKBS	O	Update clock	Boundary scan interface.
RSTCLKBS	O	Reset clock	Boundary scan interface.
SDINBS	I	Serial input data	Boundary scan interface.
SDOUTBS	O	Serial output data	Boundary scan interface.
SHCLKBS	O	Shift clock phase 1	Boundary scan interface.
SHCLK2BS	O	Shift clock phase 2	Boundary scan interface.

Name	Type	Description	Function
A	O	Addresses	Connected to MMU and LSB's to other units.
DOUT	O	Data output bus	Muxed with output of Boot-Loader, result in an internal data write bus.
DIN	I	Data input bus	Internal data read bus.
nMREQ	O	Not Memory Request	Connected to MMU. Indicates when the processor requires a memory access during the following cycle.
nRW	O	Not read write	HIGH value indicates a write cycle, LOW a read cycle of the processor. Internal read-write signal.
MAS	O	Memory Access Size	Indicates when a word transfer or a half-word or byte length is required. Valid for both read and write operations. The signals take the value 10'b for words, 01'b for half words and 00'b for bytes. Connected to the MMU unit.
BL	I	Byte Latch Control	Indicates when data and instructions are latched from the external data bus. bl(3) indicates that D(31:24) is valid, identical for bl(2) to bl(0). Generated by the MMU.
LOCK	O	Locked operation	To MMU.
ABORT	I	Memory Abort	From MMU unit and valid when the requested output is NOT allowed.

66220-92454T09

## 1.9 Interrupt Controller

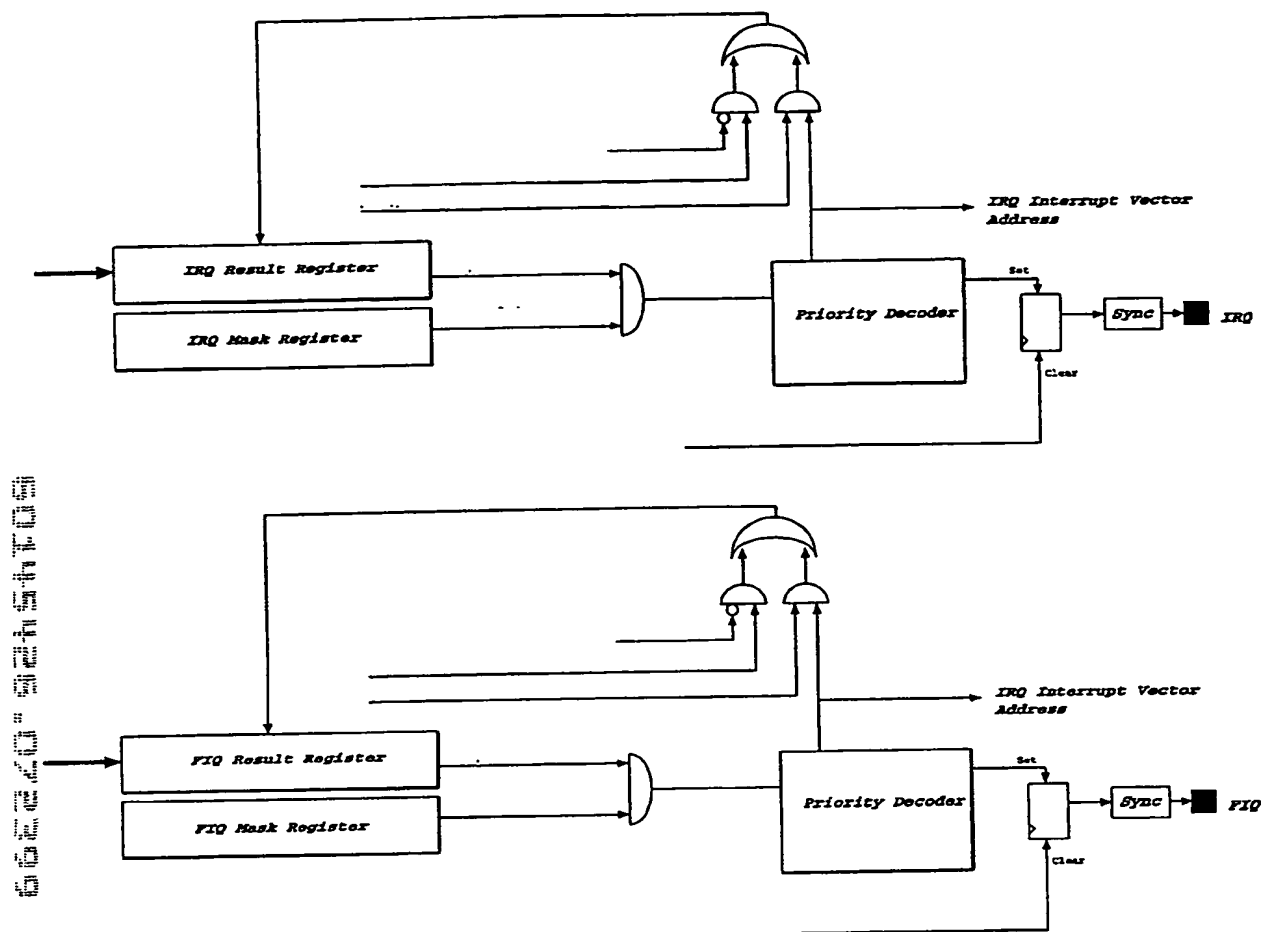


Figure 52: Interrupt controller blockdiagram

- External interrupts (TBC)
- Edge or level interrupts (TBC)
- Rising edge or falling edge interrupts (TBC)
- Synchronization of incoming interrupts (TBC)
- x events mapped to FIQ or IRQ of ARM7TDMI.
- Each event maskable via "Mask Register".
- All masked values are ORED to a single synchronized FIQ or IRQ signal.
- ARM7TDMI executes the interrupt by reading the Offset + Interrupt Vector register. This register contains the branch address of the different events. Offset adjustable.

- Polling of result register of incoming events.
- "Interrupt Vector Address" generated, depends on the priority of the events.

### 1.9.1 Interrupt Controller registers

Name	Address	Reset	RW	Function
FIQ-Mask	Interrupt Controller-address + 0x0	0x00000000	RW	Mask on Events
FIQ-Offset	Interrupt Controller-address + 0x4	0x00000000	RW	Vector Offset
FIQ-Result	Interrupt Controller-address + 0x8	0x00000000	RW	Incoming Events
FIQ-Branch	Interrupt Controller-address + 0xC	0x00000000	RW	Jump address
IRQ-Mask	Interrupt Controller-address + 0x10	0x00000000	RW	Mask on Events
IRQ-Offset	Interrupt Controller-address + 0x14	0x00000000	RW	Vector Offset
IRQ-Result	Interrupt Controller-address + 0x18	0x00000000	RW	Incoming Events
IRQ-Branch	Interrupt Controller-address + 0x1C	0x00000000	RW	Jump address

#### • Mask Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MA(31)	MA(30)	MA(29)	MA(28)	MA(27)	MA(26)	MA(25)	MA(24)	MA(23)	MA(22)	MA(21)	MA(20)	MA(19)	MA(18)	MA(17)	MA(16)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MA(15)	MA(14)	MA(13)	MA(12)	MA(11)	MA(10)	MA(9)	MA(8)	MA(7)	MA(6)	MA(5)	MA(4)	MA(3)	MA(2)	MA(1)	MA(0)

Figure 53: Mask register

Bit	Name	Reset	Function
31-0	MA	0-0	'1' = enable event, '0' = mask event

#### • Result Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RE(31)	RE(30)	RE(29)	RE(28)	RE(27)	RE(26)	RE(25)	RE(24)	RE(23)	RE(22)	RE(21)	RE(20)	RE(19)	RE(18)	RE(17)	RE(16)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RE(15)	RE(14)	RE(13)	RE(12)	RE(11)	RE(10)	RE(9)	RE(8)	RE(7)	RE(6)	RE(5)	RE(4)	RE(3)	RE(2)	RE(1)	RE(0)

Figure 54: Result register

Bit	Name	Reset	Function
31-0	RE	0-0	'1' = event valid, '0' = no event

#### • Offset Register

See figure 55

Bit	Name	Reset	Function
31-9	OF	0-0	Offset Address value

#### • Branch Register

Bit	Name	Reset	Function
3-0	-	0-0	Fixed distance between two interrupt vectors
8-4	BAD	0-0	Branch Address Decoder value
31-9	OF	0-0	Offset Address value



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OP(31)	OP(30)	OP(29)	OP(28)	OP(27)	OP(26)	OP(25)	OP(24)	OP(23)	OP(22)	OP(21)	OP(20)	OP(19)	OP(18)	OP(17)	OP(16)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP(15)	OP(14)	OP(13)	OP(12)	OP(11)	OP(10)	OP(9)	RAD(8)	RAD(7)	RAD(6)	RAD(5)	RAD(4)	RAD(3)	RAD(2)	RAD(1)	RAD(0)

Figure 55: Branch register

FIQ Event	Interrupt Vector Address	IRQ Event	Interrupt Vector Address
Event 0	FIQ offset value + 0x00	Event A	IRQ offset value + 0x00
Event 1	FIQ offset value + 0x10	Event B	IRQ offset value + 0x10
Event 2	FIQ offset value + 0x20	Event C	IRQ offset value + 0x20
...		...	
Event 31	FIQ offset value + 0x1F0	Event X	IRQ offset value + 0x1F0

### 1.9.2 Interrupt Controller principle

- **FIQ-IRQ Interrupt generation with  $Mask(i) = '1'$**  See figure 56(a).
- **Polling mechanism of events with  $Mask(i) = '0'$**  See figure 56(b).

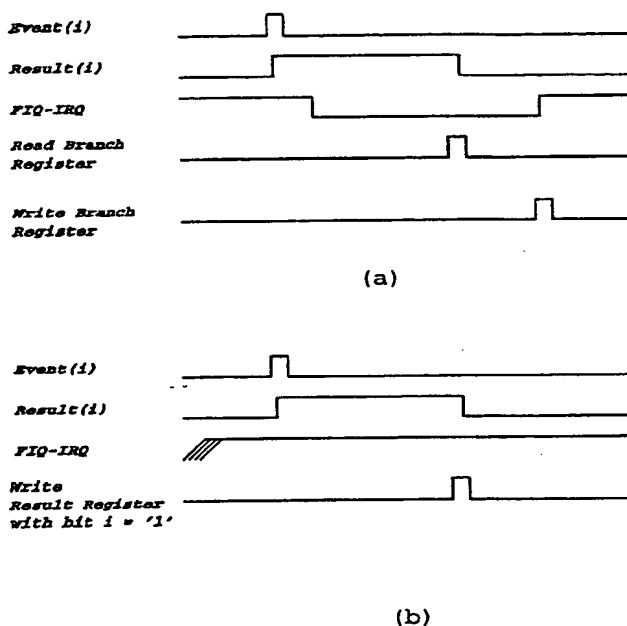


Figure 56: Interrupt principle

### 1.10 Watchdog

The aim of the watchdog is to verify the operational status of the software. Ideally, if the software is malfunctioning (e.g. crashing), the watchdog should be able to detect it and eventually reset the system. The definition of this malfunctioning can be very broad or very narrow, depending on the requirements, which could be:

- Unintentional endless loops
- Addressing of non existing memory ranges
- Access violations (e.g. of certain registers)
- Special exceptions
- Real time related problems
- ARM7TDMI malfunctioning
- ...

The watchdog is disabled after reset. Once enabled by the software, it cannot be disabled again, except by causing a reset condition. The reset cause is stored in a register, which can only be cleared by reading it out, enhancing this way the debugging of the chip.

For some of the malfunctions (access violations, ...) the watchdog can reset the system immediately. But other malfunctions (endless loops, ...) need a FSM (Finite State Machine) that must receive inputs from the software with some sequence. This could be a sequence of write actions to a register in the watchdog with varying data. For instance writing the sequence 02h, 01h, 01h, 00h, 01h, 00h, 00h, 01h. The way this writing occurs can give more or less security. Software could be functioning in the following way. On the application level of the software 02h is written to the watchdog. On the real time level the remaining sequence is written to the watchdog, 1 byte each real time interrupt. This way the FSM must not be aware of the time. Another approach could be that the watchdog is aware of time, relating it with a specific timer, that can be used by software as well. Now the correct operation can be the writing of the sequence within some timing boundaries: not too soon and not too late. TBD!

## 2 Memory map

### 2.1 Normal operation

Start Address	End Address	Block	Data bus width	Default Value	Action
0x00000000	0x000007FF	Internal SRAM	8-16-32		RW
0x10100000		SPI driver : Transmit Data	32	0x00000000	RW
0x10100004		SPI driver : Receive Data	32		R
0x10100008		SPI driver : Configuration	32	0x000063C00	RW
0x10200000		SP0 driver : Transmit Data	32	0x00000000	RW
0x10200004		SP0 driver : Receive Data	32	0x00000000	R
0x10200008		SP0 driver : Timer	32	0x00000000	RW
0x1020000C		SP0 driver : Control-Status	32	0x000001F8	RW
0x10300000		Interrupt Controller : FIQ-Mask	32	0x00000000	RW
0x10300004		Interrupt Controller : FIQ-Offset	32	0x00000000	RW
0x10300008		Interrupt Controller : FIQ-Result	32	0x00000000	RW
0x1030000C		Interrupt Controller : FIQ-Branch	32	0x00000000	RW
0x10300010		Interrupt Controller : IRQ-Mask	32	0x00000000	RW
0x10300014		Interrupt Controller : IRQ-Offset	32	0x00000000	RW
0x10300018		Interrupt Controller : IRQ-Result	32	0x00000000	RW
0x1030001C		Interrupt Controller : IRQ-Branch	32	0x00000000	RW
0x10400000		MMU CCS register	32	0x1F1F1F1F	RW
0x10400004		MMU GC register	32	0x00000003	RW
0x10500000		(S)UART : Control	32	0x00000006	RW
0x10500004		(S)UART : Mask	32	0x00000000	RW
0x10500008		(S)UART : Transmit Data	32	0x00000000	RW
0x1050000C		(S)UART : Receive Data	32	0x00000000	R
0x10500010		(S)UART : Baud rate	32	0x00000104	RW
0x10500014		(S)UART : Event-Status	32	0x00000000	RW
0x10600000		PWM-Timer : Control	32	0x00000000	RW
0x10600004		PWM-Timer : Period	32	0x00000000	RW
0x10600008		PWM-Timer : Init	32	0x00000000	RW
0x10700000		GPIO : Data	32	0x000000FF	RW
0x10700004		GPIO : Event	32	0x00000000	RW
0xC0000000	0xCFFFFFFF	CS(0)	8-16-32		RW
0xD0000000	0xDFFFFFFF	CS(1)	8-16-32		RW
0xE0000000	0xEFFFFFFF	CS(2)	8-16-32		RW
0xF0000000	0xFFFFFFF	CS(3)	8-16-32		RW

#### Remarks:

- When input signals NOBOOT is valid, ARM7TDMI core will start after reset and will fetch code from CS(0) device. This means that the memory map of the CS(0) and Internal SRAM are swapped. The CS(0) and CS(1) can be swapped by the SWA bit in the MMU control register.

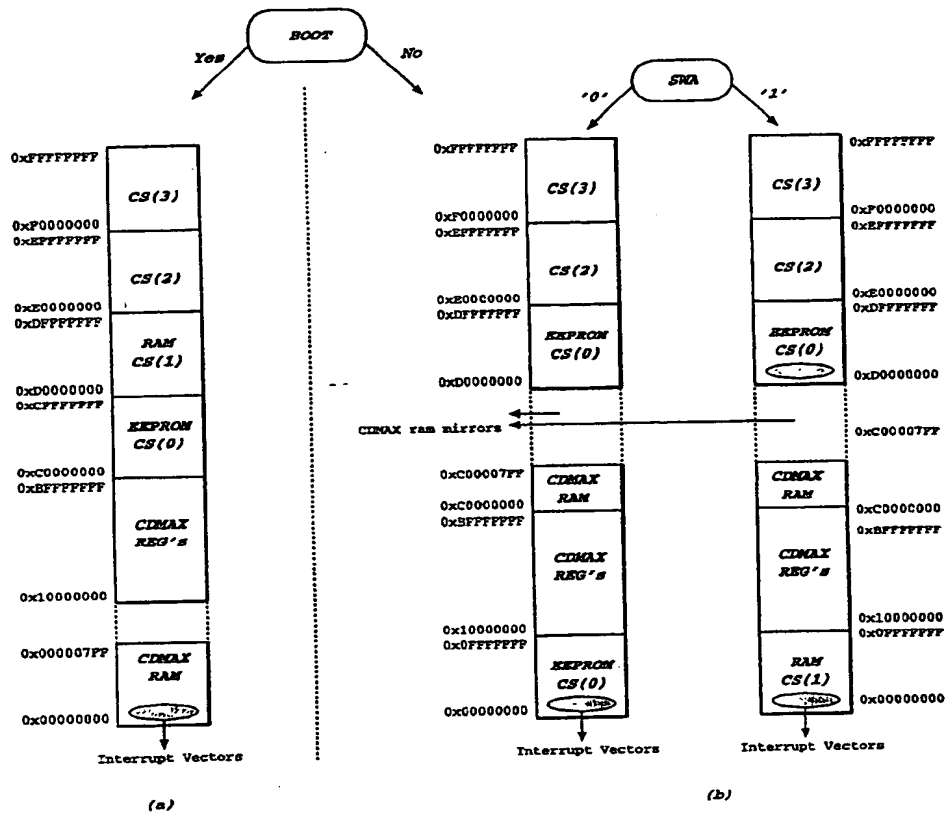


Figure 57: Internal-External memory expand mode .

### 3 ARM7TDMI timings

#### 3.1 ARM7TDMI timing

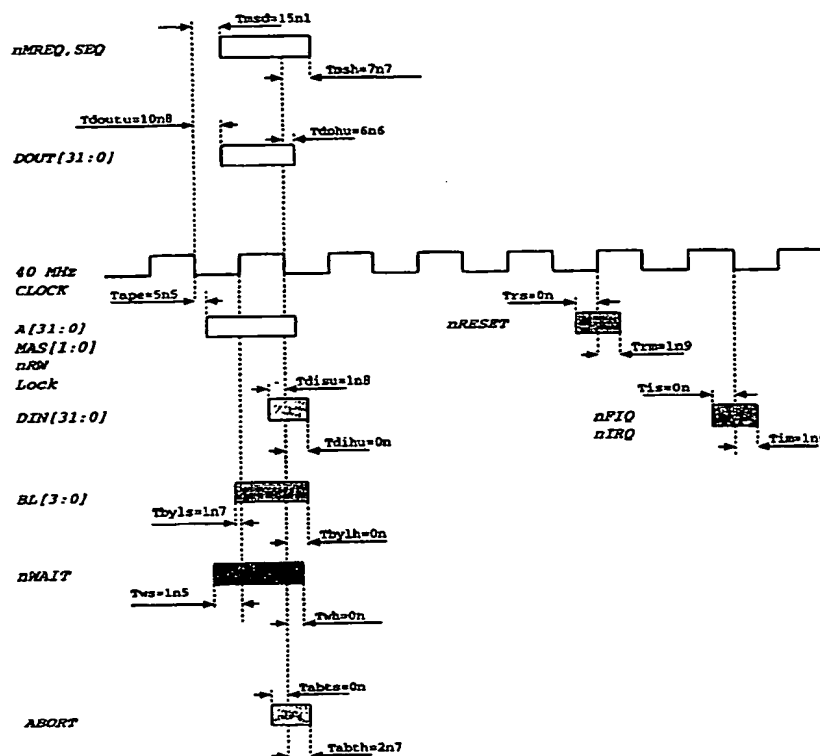


Figure 58: ARM7TDMI timing: APE = 0

#### 4 Internal bus concept

The address bus and data write bus are always driven by the ARM after booting or by the bootloader during booting. The data read bus is driven by the peripheral entities when they are selected. Otherwise a buskeeper weakly drives the bus. The peripheral entities drive the bus only when the MCLK of the ARM is at a high level. This way there is never a bus-contention problem while still meeting the timing constraints of the ARM. Care must be taken that the MCLK signal has the shortest delay to the OE-signal, which controls the tristate enables of the data read bus! The APE pin of the ARM is set to 1 (dram mode) and the (early) address of the arm is clocked into a register. The MAS signals are latched, because the setup time for a register is not met.

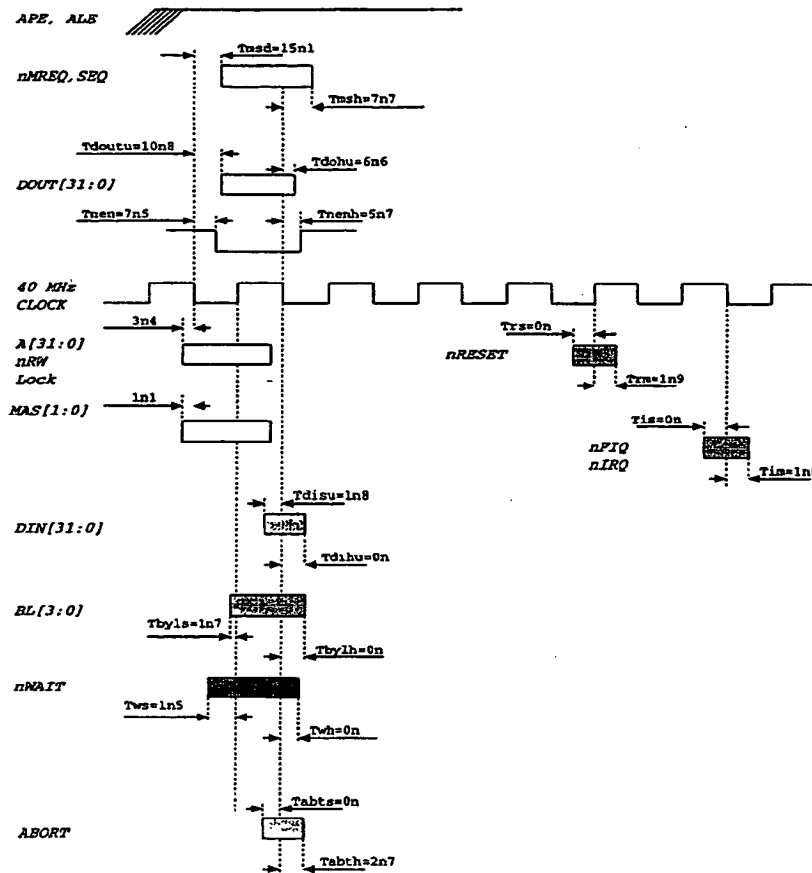


Figure 59: ARM7TDMI timing: APE = 1

## 5 Testability

Features of the JTAG interface :

- The External interface of the TAP (Test Access Port) consists of four ( add TRST) required signals.
- Tap controller controls user-defined data registers, examples of user-defined registers treated this way are internal scan chains and built-in self-test (BIST) registers.
- Optimize the routing of Boundary Scan PORTs for layout reasons.

### 5.1 ATPG test

- **Scan path** : Done by double clk flipflops OR via controlled CLK tree.
- **JTAG I/O module + TAP controller** : Use the TAP controller of ARM or implement TAP controller by synopsys. All the I/O 's should include a JTAG compatible operation ( Add vhd-code before each I/O) mode. Done by Synopsys.

Remarks :

- *Disable the OE control signals of each peripheral unit for ATPG test. Force one device on the internal 32 bit read-bus for example, the 32 bit data register of (S)UART.*

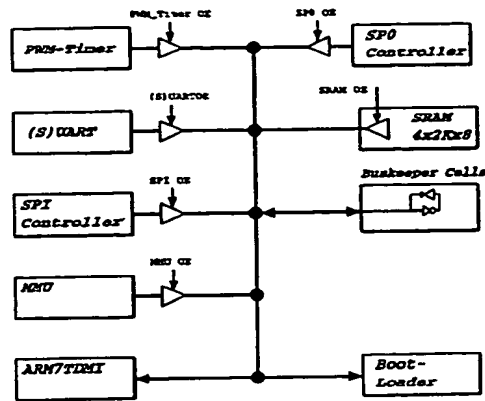


Figure 60: Internal data read bus concept

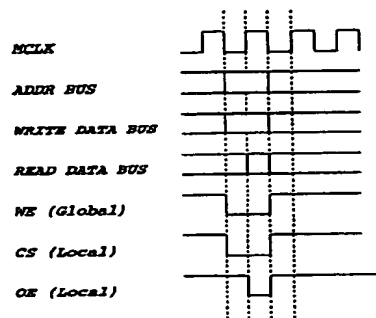


Figure 61: Internal bus timing

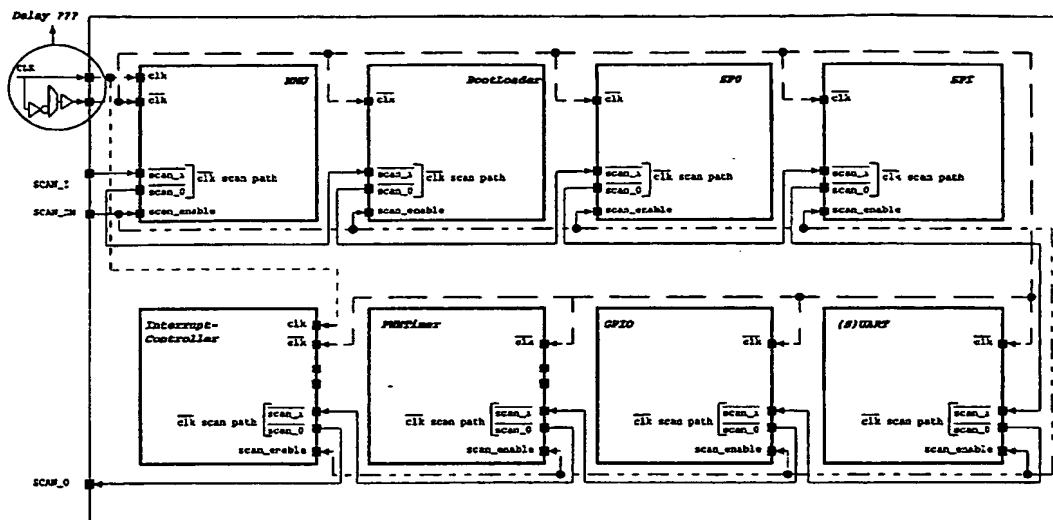


Figure 62: Peripheral scanpath

- Force to drive the internal write-bus by one master.

## 5.2 ARM debugger

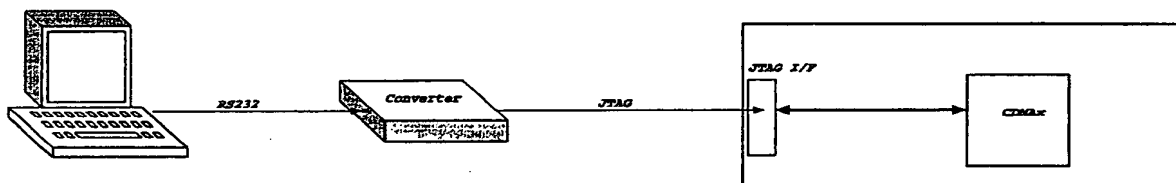


Figure 63: JTAG debug interface

- JTAG interface with hardware and software ARMSD module to set breakpoints and read/write internal ARM registers via ARM7TDMI Icebreaker module.
- "ANGEL" debug C-code. Includes debug platform, communicates with user defined access port (Example (S)UART). Link C-code with user C-code.

## 5.3 Test enable register

- Enable BIST mode via JTAG interface and user-defined data register of the TAP controller.
- All rams output fail bit mapped to a muxed output pin.
- All rams are test parallel, low test time.

## 6 Gate-Count Components





Component	Gate-Count	Max Freq	Scan FF's on INV MCLK
SPIdriver	2147	MCLK	142
SP0driver	2755	MCLK	199
PWMTimer	1852	MCLK	111
SUART	3758	MCLK	270
MemManUnit		MCLK	
BootLoader	2911	MCLK	227
GPIO	1428	MCLK	104
MuxARMBoot	457	-	-
Interrupt Controller	2894	MCLK	188
BIST Control 2048x8 SPS2	4 x	MCLK	

- vectors.s: this file contains the exception vectors
- regioninit.s: this file copies the regions of code from the load view to the execution view.

The C-file contains the actual program, which does some reading and writing to the memory map, and some simple functions. The file scat.txt contains the information for the linker, using the scatter loading principle. The reader is kindly referred to the "ARM software development toolkit user guide" for more information. The memory map for the Noboot mode depends on the swap bit in the general configuration register. If this swap is 0, then CS(0) of the flash prom is activated for two areas of the memory map (ref. 57). One of the areas is the lowest part, starting at address 0, where the ARM7TDMI fetches its first instruction. Inside the program of the ARM7TDMI (in file boots.s) a jump to the other area is done before the swap bit is set to 1. Once the swap bit is set to 1 the lowest memory area is covered by the CS(1) used for the external ram. Now parts of the program can be copied to this ram. This is done in the file regioninit.s.

The command used for the creation of the image:

```
armasm -o c:\users\mertens\boot\debug\boot.o -MD- -IC:\users\mertens\boot\sfiles
c:\users\mertens\boot\sfiles\boot.s
armasm -o c:\users\mertens\boot\debug\regioninit.o -MD- -IC:\users\mertens\boot\sfiles
c:\users\mertens\boot\sfiles\regioninit.s
armasm -o c:\users\mertens\boot\debug\vectors.o -MD- -IC:\users\mertens\boot\sfiles
c:\users\mertens\boot\sfiles\vectors.s
armcc -o c:\users\mertens\boot\debug\chandler.o -c -MD- -IC:\users\mertens\boot\cfiles
c:\users\mertens\boot\sfiles\chandler.c
armlink -o c:\users\mertens\boot\debug\bootloader.axf
c:\users\mertens\boot\debug\boot.o c:\users\mertens\boot\debug\regioninit.o
c:\users\mertens\boot\debug\vectors.o c:\users\mertens\boot\debug\chandler.o
```

The result of the linker is a xxx.axf file. From this file a xxx.i32 file, which is a intel hex format file, is generated with the command:

```
c:\arm250\bin\fromelf -nozerpath -nodebug -bootloader.axf -i32 cdmax.i32
```

This cdmax.i32 file is used in the simulation testbench immediately, provided the simulation is done with the entity: "New\_EEprom.i32.vhd".

### 7.1.2 Boot mode program

All files are located on PC named "Friday", in the (sub)-directory : c:/users/mertens/cdmax. The listings are included the appendices. There is one assembly language file (init.s) which initialises the exception vectors and jumps to the main program. The main program is a C-language file (chandler.c) and it executes some reading and writing to the memory map, and some simple functions.

The command used for the creation of the image:

```
armasm -o c:\users\mertens\boot\cdmax\init.o -MD- -IC:\users\mertens\cdmax\sfiles
c:\users\mertens\cdmax\sfiles\init.s
```

```
armlink -o c:\users\mertens\cdmax\debug\cdmax.axf
c:\users\mertens\cdmax\debug\chandler.o
c:\users\mertens\cdmax\debug\init.o
```

The result of the linker is a xxx.axf file. From this file a xxx.ihf file, which is a intel hex format file, is generated with the command:

```
c:\arm250\bin\fromelf -nozerpath -nodebug -boatloader.axf -ihf cdmax.ihf
```

This cdmax.ihf file is used in the simulation testbench immediately, provided the simulation is done with the entity: "New\_EEPROM.vhd". Now a data conversion is done: Data convert : Copy file cdmax.ihf from PC name "FRIDAY", directory : c:/users/mertens/CDMAX/debug to workstation on directory /users/mertens/designs/Chips/CDMAX/Testbench/C-Code/ARM-Code/Test1 and run "convnboot" when the ARM7TDMI will start with the NOBOOT options enabled. Convert with "convboot" when normal boot-loader principle is used. The testbench must use the entity "New\_EEPROM.vhd".

The executable under SPARC environment: sun\_tekhex script : Used by the convnboot and convboot routines.

usage: tekhex [options]

Converts a Tektronix Hex file into the TI-DSP download format

Version 0.3 dd 15 mar 1999 Jan Mennekens Sirius Communications

The following command line options are supported :

- i file input file (default : stdin)
- o file output file (default : stdout)
- I ID boot ID [0..255] (default : 0)
- a address address of command (default : 0)
- c command put command at address (default : none)
- h don't output headers (default : output headers)
- T transform endianness (default : no change)  
(assumes 32-bit words) (default : no change)
- w addresses are in words (default : byte addresses)
- l output in little-endian (default : big endian)
- v verbose mode (default : silent)

662220:0245103

APPENDICES**A Boot.s**

```

; 1.1.2.2
;*****
;*      File: boot.s                      *
;*  Purpose: Application Startup Code      *
;*****
;
; This code performs all the initialization required before
; branching to the main C application code. It defines the
; ENTRY point, initializes the Stack Pointers for each mode,
; copies RO code and RW data from ROM to RAM and zero-initializes
; the ZI data areas used by the C code.
;
; This startup code is intended for use with hardware such as
; the ARM PID board, where memory management maps an aliased
; copy of the ROM at 0x04000000 to address zero on reset.
; Following reset, RAM is mapped into address zero, and the
; code then branches to execute from the real ROM.

                AREA    Boot, CODE, READONLY

; Some standard definitions...

Mode_USR       EQU      0x10
Mode_FIQ       EQU      0x11
Mode_IRQ       EQU      0x12
Mode_SVC       EQU      0x13
Mode_ABT       EQU      0x17
Mode_UNDEF     EQU      0x1B
Mode_SYS       EQU      0x1F                ; only available on ARM Arch. v4

I_Bit          EQU      0x80
F_Bit          EQU      0x40

Ext_Ram_Base EQU 0xE0000800 ; SWA=0 and boot=false
Ext_Ram_Limit EQU 0xC0000700 ; Idem

SET_SWAP EQU 0x00000004 ;
SWAP_ADDR EQU 0x10400004 ;

SET_CCS EQU 0x13425012 ;
CCS_ADDR EQU 0x10400000 ;

SVC_Stack EQU Ext_Ram_Limit
USR_Stack EQU Ext_Ram_Limit-128
FIQ_Stack EQU Ext_Ram_Limit-256

ROM_Start      EQU      0xD0000000                ; Base address of ROM after remapping
Instruct_2     EQU      ROM_Start + 4              ; Address of second instruction in ROM

ResetBase      EQU      0x0B000000                ; Address of reset controller base

```

```
ClearResetMap EQU ResetBase + 0x20 ; Offset of remap control from base
```

```
; --- Define entry point
ENTRY
```

```
Start_Boot
EXPORT Start_Boot
```

```
; --- Continue execution from ROM rather than aliased copy at zero
LDR pc, =Instruct_2
```

```
; --- Flip the memory map by writing to the ClearResetMap location
; in the Reset and Pause Controller
```

```
LDR r1, =CCS_ADDR
LDR r0, =SET_CCS
STR r0, [r1]
LDR r1, =SWAP_ADDR
LDR r0, =SET_SWAP
STR r0, [r1]
```

```
; --- Initialize stack pointer registers
; Enter IRQ mode and set up the IRQ stack pointer
; MOV r0, #Mode_IRQ:OR:I_Bit:OR:F_Bit ; No interrupts
; MSR cpsr_c, r0
; LDR sp, =IRQ_Stack
```

```
; Enter FIQ mode and set up the FIQ stack pointer
; MOV r0, #Mode_FIQ:OR:I_Bit:OR:F_Bit ; No interrupts
; MSR cpsr_c, r0
; LDR sp, =FIQ_Stack
```

```
; Enter ABT mode and set up the ABT stack pointer
; MOV r0, #Mode_ABT:OR:I_Bit:OR:F_Bit ; No interrupts
; MSR cpsr_c, r0
; LDR sp, =ABT_Stack
```

```
; Enter IRQ mode and set up the IRQ stack pointer
; MOV r0, #Mode_UNDEF:OR:I_Bit:OR:F_Bit ; No interrupts
; MSR cpsr_c, r0
; LDR sp, =UNDEF_Stack
```

```
; Set up the SVC stack pointer last and return to SVC mode
MOV r0, #Mode_SVC:OR:I_Bit:OR:F_Bit ; No interrupts
MSR cpsr_c, r0
LDR sp, =SVC_Stack
```

```
; --- Initialize memory system
; ...
```

```
; --- Initialize critical IO devices
; ...
```

```
; --- Initialize interrupt system variables here
```

```

; ...

; --- Initialize memory required by main C code

    IMPORT InitRegions
    BL     InitRegions ; in regioninit.s

; --- Now enable IRQs, change to user mode and set up user mode stack.

;     MOV     r0, #Mode_USR:OR:F_Bit ; IRQ enabled
;     MSR     cpsr_c, r0
;     LDR     sp, =USR_Stack

; --- Now we enter the main C application code

    IMPORT c_init
; If the main C code is in Thumb code rather than ARM,
; we would need to change to Thumb state here.

;     BL      c_init ;
;     LDR     pc,=c_init ;

; If above subroutine ever returns, just sit in an endless loop
here    B      here

    END

```

## B Regioninit.s

```

;*****
;*      File: regioninit.s                      *
;*  Purpose: Application Startup Code            *
;*****
;
; This file contains the macro and supporting subroutines to
; copy RO code and RW data from ROM to RAM and zero-initialize
; the ZI data areas in RAM.

    AREA RegionInit, CODE, READONLY

    EXPORT InitRegions

; This macro:
; a) copies RO code and/or RW data from ROM at Load$$area$$Base
; to RAM at Image$$area$$Base, of length Image$$area$$Length bytes.
; b) fills with zero the ZI data in RAM at Image$$area$$ZI$$Base,
; of length Image$$area$$ZI$$Length bytes.

    MACRO
        macro_RegionInit $areaname

```

```

LCLS namecp
LCLS copyloadsym
LCLS copybasesym
LCLS copylensym
; LCLS zibasesym
; LCLS zilensym

```

```
namecp SETS "$areaname"
```

```

copyloadsym SETS "|Load$$":CC:namecp:CC:"$$Base|"
copybasesym SETS "|Image$$":CC:namecp:CC:"$$Base|"
copylensym SETS "|Image$$":CC:namecp:CC:"$$Length|"
;zibasesym SETS "|Image$$":CC:namecp:CC:"$$ZI$$Base|"
;zilensym SETS "|Image$$":CC:namecp:CC:"$$ZI$$Length|"

```

```

; The following symbols are generated by the linker. They are imported
; WEAKly because they may not all have defined values. Those which are
; undefined will take the value zero.

```

```

IMPORT $copyloadsym, WEAK
IMPORT $copybasesym, WEAK
IMPORT $copylensym, WEAK
;
;
IMPORT $zibasesym, WEAK
IMPORT $zilensym, WEAK

```

```

LDR r0,=$copyloadsym ; load address of region
LDR r1,=$copybasesym ; execution address of region
MOV r2, r1 ; copy execution address into r2
LDR r4,=$copylensym
ADD r2, r2, r4 ; add region length to execution address to...
; ...calculate address of word beyond end...
; ... of execution region

```

```
BL copy
```

```

; LDR r2,=$zilensym ; get length of ZI region
; LDR r0,=$zibasesym ; load base address of ZI region
; MOV r1, r0 ; copy base address of ZI region into r1
; ADD r1, r1, r2 ; add region length to base address to...
; ...calculate address of word beyond end...
; ... of ZI region

```

```
; BL zi_init
```

```
MEND
```

```

; InitRegions is called from boot.s to initialize the specified execution regions.
; In this example, the regions are called 32bitRAM and 16bitRAM.
; These execution region names should match those given in the scatter description file.

```

```

InitRegions
    STMFD sp!,{lr}
    macro_RegionInit 32bitRAM
    macro_RegionInit programma
    LDMFD sp!,{pc}

```

; --- copy and zi\_init subroutines

; copy is a subroutine which copies a region, from an address given by  
; r0 to an address given by r1. The address of the word beyond the end  
; of this region is held in r2. r3 is used to hold the word being copied.  
copy

```

CMP    r1, r2          ; loop whilst r1 < r2
LDRLO  r3, [r0], #4
STRLO  r3, [r1], #4
BLO    copy
MOV     pc, lr          ; return from subroutine copy

```

; zi\_init is a subroutine which zero-initialises a region,  
; starting at the address in r0. The address of the word  
; beyond the end of this region is held in r1.  
zi\_init

```

MOV     r2, #0
CMP     r0, r1          ; loop whilst r0 < r1
STRLO  r2, [r0], #4
BLO     zi_init
MOV     pc, lr          ; return from subroutine zi_init

END

```

## C Vectors.s

AREA Vect, CODE, READONLY

; These exception vectors and dummy exception handlers will be  
; relocated at start-up from FLASH to 32bitRAM at address 0

; \*\*\*\*\*  
; Exception Vectors  
; \*\*\*\*\*

```

LDR     PC, Reset_Addr
LDR     PC, Undefined_Addr
LDR     PC, SWI_Addr
LDR     PC, Prefetch_Addr
LDR     PC, Abort_Addr
NOP                                           ; Reserved vector
LDR     PC, IRQ_Addr
LDR     PC, FIQ_Addr

```

IMPORT Start\_Boot ; In boot.s

```

Reset_Addr    DCD    Start_Boot
Undefined_Addr DCD    Undefined_Handler
SWI_Addr      DCD    SWI_Handler
Prefetch_Addr DCD    Prefetch_Handler
Abort_Addr    DCD    Abort_Handler

```



```

                DCD      0                ; Reserved vector
IRQ_Addr        DCD      IRQ_Handler
FIQ_Addr        DCD      FIQ_Handler

```

```

; *****
; Dummy Exception Handlers
; *****
; The following handlers do not do anything useful in this example.
; They are set up here for completeness.

```

```

Undefined_Handler
    B        Undefined_Handler
SWI_Handler
    B        SWI_Handler
Prefetch_Handler
    B        Prefetch_Handler
Abort_Handler
    B        Abort_Handler
IRQ_Handler
    B        IRQ_Handler
FIQ_Handler
    B        FIQ_Handler

    END

```

## D Chandler.c

## E Init.s

```

; -----
; Assembler code handles the exceptions
; -----

```

```

        AREA    Init, CODE, READONLY

```

```

; -----
; Memory Map locations and constants
; -----

```

```

MODE_SVC EQU 0x13
MODE_FIQ EQU 0x11
MODE_USR EQU 0x10

```

```

I_Bit EQU 0x80
F_Bit EQU 0x40

```

```

Ext_Ram_Base EQU 0xD0000800 ; SWA=0 and boot=false
Ext_Ram_Limit EQU 0xE0000800 ; Idem

```

```
SVC_Stack EQU Ext_Ram_Limit
USR_Stack EQU Ext_Ram_Limit-128
FIQ_Stack EQU Ext_Ram_Limit-256
```

```
; -----
; Define entry point
; -----
EXPORT __main
__main
ENTRY
```

```
B    Reset_Handler
B    Undefined_Handler
B    SWI_Handler
B    Prefetch_Handler
B    Abort_Handler
NOP      ; Reserved vector
B    IRQ_Handler
B    FIQ_Handler
```

```
;
Undefined_Handler
    B Undefined_Handler
SWI_Handler
    B SWI_Handler
Prefetch_Handler
    B Prefetch_Handler
Abort_Handler
    B Abort_Handler
IRQ_Handler
    B IRQ_Handler
FIQ_Handler
    B FIQ_Handler
```

```
; -----
; The Reset_handler routine
; -----
```

```
Reset_Handler
```

```
; ** Enter IRQ mode and set up the IRQ stack pointer, first select svc mode and
; ** enable the FIQ
```

```
; MOV RO,#MODE_USR:OR:I_Bit:OR:F_Bit ; 0x10 user mode
; MSR CPSR_c,RO
; LDR R13, =USR_Stack
```

```
; MOV RO,#MODE_FIQ:OR:I_Bit:OR:F_Bit ; 0x11 fiq mode
; MSR CPSR_c,RO
; LDR R13, =FIQ_Stack
```

```
MOV RO,#MODE_SVC:OR:I_Bit:OR:F_Bit ; 0x13 supervisor mode
MSR CPSR_c,RO
LDR R13, =SVC_Stack
```

```
; ** Enter the C code
```

```
IMPORT c_init
MOV RO,RO
BL c_init

END
```

60145420-072299

---

# CDMAx

CDMAx specification

Status: V9-n  
Doc no CDMAx-1  
July 12, 1999  
Nico Lugil



## Contents

1	Introduction	7
2	Abbreviations and conventions	7
2.1	Abbreviations	7
2.2	Conventions	7
3	General definitions	8
3.1	Clock definitions	8
3.2	Transmitter clocks	8
3.3	Receiver clocks	9
4	Transmitter parameter programming/interface with ARM	10
5	Transmitter Specification	11
5.1	Gold code module	12
5.2	QPN channels with synchronisation hardware and PNcode generators	15
5.2.1	Synchronisation hardware with chip reslution	15
5.2.2	QPN channel sets	16
5.2.3	PNcode generators	23
5.2.4	Synchronicity summary	34
5.3	Burst generator	34
5.4	Combiners A, B and C	35
5.5	Scrambler and scrambling code generator	35
5.5.1	Scrambling code generator	35
5.5.2	Scrambler	37
5.6	Combiner 3	38
5.7	Interpolator with chip phase control	38
5.8	Upsampling4 and programmable filter	39
5.9	Offset modulation	41
5.10	Hold 1-1024	41
5.11	Complex upconverter and NCO	42
5.11.1	NCO	42
5.11.2	Upconverter	43
5.12	Level Control part 1	44
5.13	Phase angle compensation	45
5.14	Level Control part 2	46
5.15	Transmitter output block	47
5.16	Asynchronous TX synchronisation	48
5.17	Global reset	48
5.18	Transmitter parameters	49
6	Receiver Specification	56
6.1	Input sampling and selection	57
6.2	DC removal filter	57
6.3	Common downconverter with NCO	57
6.3.1	NCO	57
6.3.2	Downconverter	58
6.4	Truncation DC removal	58

6.5	Decimation 1-1024	59
6.6	Level Control 1	60
6.7	Programmable 49 taps FIR filter with programmable downsampling	61
6.8	Level control 2	62
6.9	Noise - DC offset estimator	63
6.10	Acquisition hardware	64
6.11	Demodulators 0 and 1	64
7	Acquisition hardware	65
8	Demodulators 0 and 1 structure	66
8.1	Tracking unit Type 0 structure	68
8.2	Tracking unit Type 1 structure	69
9	Demodulator building blocks	70
9.1	Downconverter and NCO	70
9.1.1	Tracking unit NCO	70
9.1.2	Tracking unit Downconverter	71
9.2	Tracking unit Interpolator	72
9.2.1	General interpolator principle	72
9.2.2	Tracking unit interpolator operation	75
9.3	MEL gate	76
9.4	Downsampling factor 2	76
9.5	chipstream selection	77
9.6	Scrambling code generator and descramblers	78
9.6.1	Tracking unit scrambling code generator	78
9.6.2	Descrambler	78
9.7	Tracking units Despreaders	80
9.7.1	Tracking units Despreaders overview	80
9.7.2	Despreaders detailed description	81
9.8	Variable amplifiers	82
9.9	Channel estimation	82
9.10	Channel correction	82
9.11	PNcode generators	83
9.12	AED and AGC	83
9.13	PED and PLL	83
9.14	TED and DLL	83
9.15	symbol combiner	83
9.16	CCP - Chip Combining Part	83
10	Demodulator loops	84
10.1	Carrier phase/frequency tracking	84
10.2	Chip phase/frequency tracking	84
10.3	Symbol amplitude tracking	84
11	Demodulator configured to track 3 QPN sources	85
12	Demodulator configured to track 3 OQPN sources	87
13	Demodulator as CCMR (Chip Combining Multipath Receiver)	89

**CONTENTS**

<b>14 CCP overview</b>	<b>91</b>
14.1 CCP finger	93
14.1.1 Descrambler	93
14.1.2 Complex pilot despreader	93
14.1.3 Variable Amplifier	94
14.1.4 Slotwise coherent pilot symbol accumulation	94
14.1.5 Finger energy calculation	95
14.1.6 Channel estimator	95
14.1.7 Channel correction	98
14.1.8 Zero forcing	98
<b>15 Applications</b>	<b>98</b>
15.1 UMTS uplink	99
15.1.1 Transmitter configuration	99
15.1.2 Receiver configuration	99
15.2 UMTS downlink	99
15.2.1 Transmitter configuration	99
15.2.2 Receiver configuration	99
<b>16 Receiver wordlengths argumentation DRAFT</b>	<b>100</b>

66E220-92H54T09

## List of Figures

1	<i>TX parameter programming/interface with ARM</i>	10
2	<i>Global TX structure</i>	11
3	<i>Gold code module</i>	12
4	<i>Gold code (re)start</i>	13
5	<i>Gold code generator signals</i>	13
6	<i>Channel set synchronisation hardware</i>	15
7	<i>QPN channel set</i>	16
8	<i>QPN channel structure</i>	16
9	<i>Differential modulation block</i>	17
10	<i>Spreaders timing</i>	19
11	<i>Phasestep - Delay</i>	20
12	<i>Phasestep - Advance</i>	21
13	<i>PNcode generator for set A</i>	23
14	<i>PNcode generator for set B</i>	23
15	<i>PNcode generator for set C</i>	24
16	<i>RAM based codegenerator</i>	25
17	<i>address generator RAM</i>	26
18	<i>time relation between sync, start and symbol edge.</i>	26
19	<i>possible RAM configuration, example 1</i>	28
20	<i>possible RAM configuration, example 2</i>	29
21	<i>possible RAM configuration, example 3</i>	30
22	<i>possible RAM configuration, example 4</i>	31
23	<i>possible RAM configuration, example 5</i>	32
24	<i>possible RAM configuration, example 6</i>	33
25	<i>Mutual synchronicity</i>	34
26	<i>Scrambler with scrambling code generator</i>	35
27	<i>Scrambling code generator start</i>	35
28	<i>Interpolator with chip phase control</i>	38
29	<i>Upsampling<sub>4</sub> and programmable filter</i>	39
30	<i>offset modulation</i>	41
31	<i>Upsampling-Hold</i>	41
32	<i>NCO with frequency and phase control</i>	42
33	<i>Upconverter</i>	43
34	<i>Level control part 1</i>	44
35	<i>Phase angle compensation</i>	45
36	<i>Level control part 2</i>	46
37	<i>Transmitter output</i>	47
38	<i>Asynchronous TX synchronisation signal</i>	48
39	<i>Asynchronous TX synchronisation effect</i>	48
40	<i>Asynchronous reset</i>	48
41	<i>Global RX structure</i>	56
42	<i>common NCO with frequency, phase and phasestep control</i>	57
43	<i>Common RX downconverter</i>	58
44	<i>MAx-C-Mizer decimation 1-1024 phase timing</i>	59
45	<i>RX common level control</i>	60
46	<i>Programmable filter and downsampling</i>	61
47	<i>MAx-C-Mizer progr. 49 taps filter subsampling phase timing</i>	62



## LIST OF FIGURES

48	<i>RX common level control</i>	62
49	<i>Noise - DC offset demodulator</i>	63
50	<i>Demodulator overview</i>	67
51	<i>Tracking unit Type 0 structure</i>	68
52	<i>Tracking unit Type 1 structure</i>	69
53	<i>Tracking unit NCO and downconverter position</i>	70
54	<i>Tracking unit NCO</i>	71
55	<i>Tracking unit Interpolator</i>	72
56	<i>MEL gate</i>	76
57	<i>Downsampling factor 2</i>	76
58	<i>MAz-C-Mizer Downsampling factor 2 phase timing</i>	77
59	<i>Tracking unit chipstream selection</i>	77
60	<i>Tracking unit scrambling code generator and descramblers</i>	78
61	<i>Tracking unit descrambler</i>	78
62	<i>Tracking unit type 0 depredators overview</i>	80
63	<i>Tracking unit type 1 depredators overview</i>	80
64	<i>Tracking unit Depredators</i>	81
65	<i>MAz-C-Mizer despreaders phase timing</i>	82
66	<i>Demodulator used to track 3 QPN sources</i>	85
67	<i>Demodulator used to track 3 OQPN sources</i>	87
68	<i>Demodulator used as CCMR</i>	89
69	<i>CCP overview</i>	91
70	<i>Correlators synchronisation with the incoming chipstream</i>	92
71	<i>one CCP finger</i>	93
72	<i>Slotwise coherent pilot symbol accumulation</i>	94
73	<i>Finger energy calculation</i>	95
74	<i>Slow channel estimation filter</i>	96
75	<i>CCP finger process in the case of Slow fading mode</i>	96
76	<i>Interpolation for fast fading mode</i>	97
77	<i>CCP finger process in the case of Fast fading mode</i>	98

© 1999 by Sirius Communications NV. All rights reserved. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Sirius Communications NV. The information of this document is subject to change without notice. Sirius Communications NV shall not be responsible for any errors that may appear in this document. Sirius Communications NV makes no commitment to update or keep current the information contained in this document. Devices sold by Sirius Communications NV are covered by warranty and patent indemnification provisions appearing in Sirius Communications NV Terms and Conditions of Sale only.

60145426-072299

## 1 Introduction

This document is a collection of functional requirements/specifications for the CDMAx ASIC.  
Wordlengths are TBC and MUST be reviewed !






## 2 Abbreviations and conventions

### 2.1 Abbreviations

BS	Base station
MS	Mobile station
SF	Spreading factor
DPCCH	Dedicated Physical Channel
DPCCH	Dedicated Physical Control Channel
DPDCH	Dedicated Physical Data Channel
PRACH	Physical Random Access Channel
CCPCH	Common Control Physical Channel
UL	Uplink
DL	Downlink
HO	HandOver
CCMR	Chip Combining Multipath Receiver
MRC	Maximum Ratio Combining
CCP	Chip Combining Part

### 2.2 Conventions

The following conventions apply in figures :

	real signal or 1 branch of a complex signal
	complex signal
	collection of signals (eg control signals)
	programmable/observable setup parameter or slow runtime (ARM) parameter
	high-speed runtime parameter (data, activity, ...). High speed interfaces or firmware

The setup parameters in this document are the parameters without interface.

### 3 General definitions

#### 3.1 Clock definitions

This section contains the definition and description of the different clocks used inside the CDMAx ASIC.

#### 3.2 Transmitter clocks

clock name	description	max f.	equidist?	remarks
$fs_{A0}^I$	Transmit set A, channel 0, I spreader symbolrate	20 MHz	•	
$fs_{A1}^I$	Transmit set A, channel 1, I spreader symbolrate	20 MHz	•	
$fs_{A2}^I$	Transmit set A, channel 2, I spreader symbolrate	20 MHz	•	
$fs_{A3}^I$	Transmit set A, channel 3, I spreader symbolrate	20 MHz	•	
$fs_{A0}^Q$	Transmit set A, channel 0, Q spreader symbolrate	20 MHz	•	
$fs_{A1}^Q$	Transmit set A, channel 1, Q spreader symbolrate	20 MHz	•	
$fs_{A2}^Q$	Transmit set A, channel 2, Q spreader symbolrate	20 MHz	•	
$fs_{A3}^Q$	Transmit set A, channel 3, Q spreader symbolrate	20 MHz	•	
$fs_{B0}^I$	Transmit set B, channel 0, I spreader symbolrate	20 MHz	•	
$fs_{B1}^I$	Transmit set B, channel 1, I spreader symbolrate	20 MHz	•	
$fs_{B2}^I$	Transmit set B, channel 2, I spreader symbolrate	20 MHz	•	
$fs_{B3}^I$	Transmit set B, channel 3, I spreader symbolrate	20 MHz	•	
$fs_{B0}^Q$	Transmit set B, channel 0, Q spreader symbolrate	20 MHz	•	
$fs_{B1}^Q$	Transmit set B, channel 1, Q spreader symbolrate	20 MHz	•	
$fs_{B2}^Q$	Transmit set B, channel 2, Q spreader symbolrate	20 MHz	•	
$fs_{B3}^Q$	Transmit set B, channel 3, Q spreader symbolrate	20 MHz	•	
$fs_{C0}^I$	Transmit set C, channel 0, I spreader symbolrate	20 MHz	•	
$fs_{C1}^I$	Transmit set C, channel 0, Q spreader symbolrate	20 MHz	•	
$fc$	chiprate	20 MHz	•	
$f4c$	4 times oversampled chiprate	80 MHz	•	
$fo$	oversampled and Hold, TX output rate.	80 MHz	•	

The highest clock used in the transmit chain ( $fo$ ) should be available outside the chip.

## 3.3 Receiver clocks

Incomplete !!

clock name	description	max f.	equidist?	remarks
<i>fri</i>	Receiver input clock	80 MHz	•	
<i>frd</i>	Receiver decim output clock	80 MHz	•	
<i>fr2ct</i>	RX decim filter output	40 MHz	•	
<i>fcs<sub>D0</sub><sup>T0</sup></i>	Despreaders input rate (secondary chiprate)	40 MHz		nominal=max/2
<i>fcs<sub>D1</sub><sup>T1</sup></i>	Despreaders input rate (secondary chiprate) tracking unit 2, demod 1	40 MHz		nominal=max/2

001451426-072299

## 4 Transmitter parameter programming/interface with ARM

This section gives a short overview on how different parameters can be programmed by the ARM. Only a functional description is given. For more details and information, see SL.

The global structure of the interface between the ARM and the datapath parameters is the following :

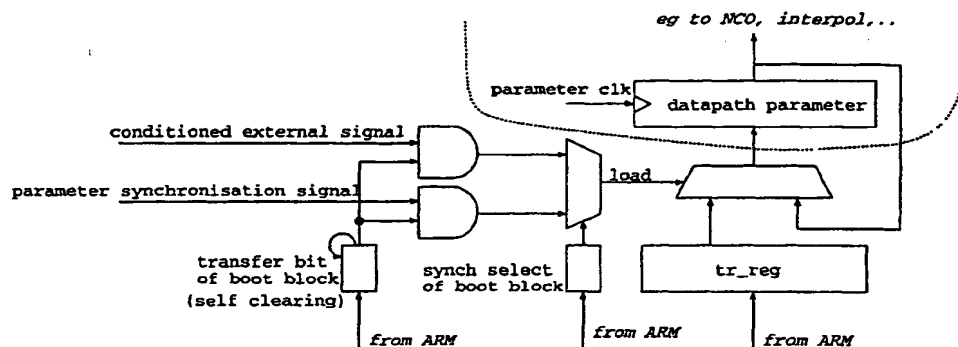


Figure 1: TX parameter programming/interface with ARM

Things like clock synchronisation, edge detection, etc are not shown on this figure, only functional description.

The ARM writes the parameters in the tr\_reg registers, this value can be transferred to the datapath synchronous with a 'parameter synchronisation signal' or with a 'conditioned external signal'. This transfer is however only done when the ARM has set the 'transfer bit of the boot block'. When the transfer is done, the transfer bit is auto-cleared. With the 'synch select of the boot block' we can select either the 'parameter synchronisation signal' or the 'conditioned external signal'.

The 'conditioned external signal' is a TBD signal (synchronisation, edge detection, ...).

The transmitter is divided in 6 boot blocks, each has one 'transfer bit' and one 'synch select' signal. The 'conditioned external signal' is common for all boot blocks.

The 'parameter synchronisation signal' is a signal like a symbol clock.

The transfer of the datapath parameter must be latency (functional and pipelining) compensated.

Some parameters have an extra transfer control so that the parameter is only transferred when the ARM wrote a new value to the tr\_reg register.

In the following chapters the signals

signal

in the figures are the datapath parameters.

See section 5.18 for an overview of all transmitter parameters.

## 5 Transmitter Specification

The global transmitter structure is shown in fig 2

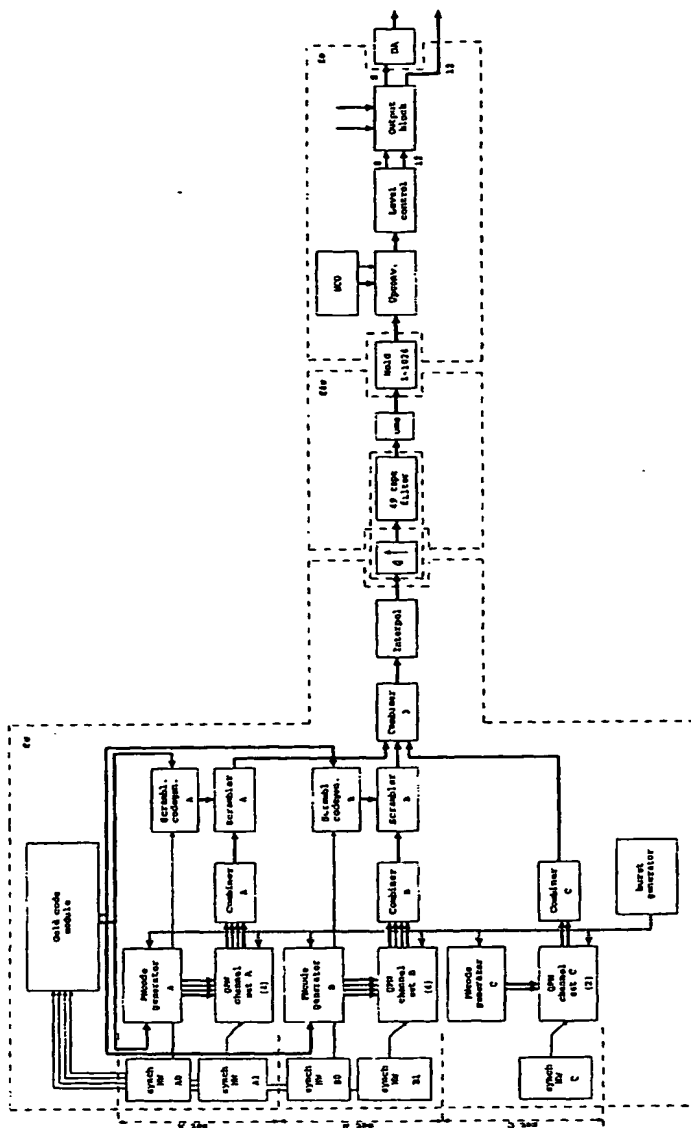


Figure 2: Global TX structure

## 5.1 Gold code module

See fig 3

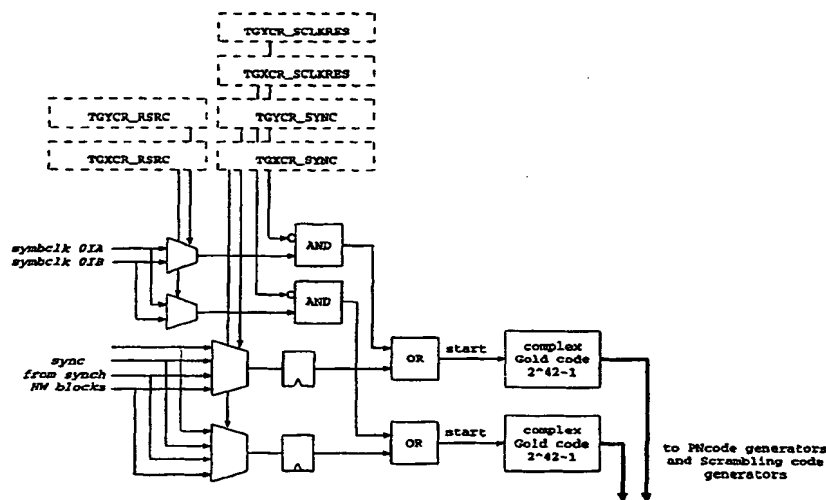


Figure 3: Gold code module

The Gold code module contains 2 complex Gold code generators. Everything runs at  $f_c$  rate. The outputs of these Gold code generators go to PNcode generators and Scrambling code generators. In this way each of the Gold code generators can be used as input of each of the 4 code generators.

There are 2 ways to (re)start the Gold code generators : by one of the 4 sync signals of set A and B, or by the symbol clocks of branch 10 of set A and B. With the TG.CR\_SYNC inputs the correct sync signal can be selected. With TG.CR\_RSRC the correct symbol clock can be selected. With TG.CR\_SCLKRES the symbolclk control can be set on or off.

TG.CR_RSRC	reference symbol clk
0	symb clk 0IA
1	symb clk 0IB

TG.CR_SYNC	synch HW
00	A0
01	A1
10	B0
11	B1

TG.CR_SCLKRES	symbol clock restart
0	ON
1	OFF



Fig 4 gives the timing relation between sync pulse, symbolclk pulse, start pulse and Gold code start.

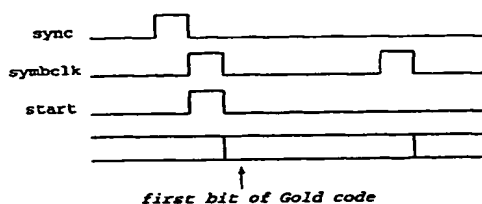


Figure 4: Gold code (re)start

The complex Gold code generator as such consists of 2 real classical Gold code generators with 42-bit registers which can generate any Gold code with any length upto  $2^{42} - 1$ . It can also be used to generate any segment out of a Gold code smaller than  $2^{42} - 1$ . Fig 5 gives an overview of the input signals of one complex Gold code generator.

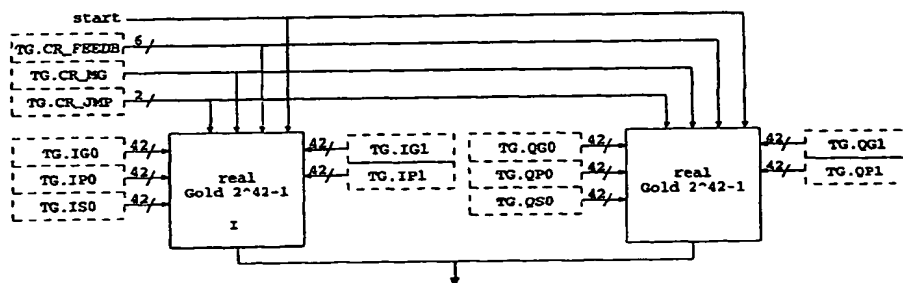


Figure 5: Gold code generator signals

Bit TG.CR\_MG selects between Gold or M codes. When M code is selected only one of the 2 M code generators in a Gold code generator is used.

TG.CR_MG	
0	Gold code
1	M-code

The TG.CR\_FEEDB inputs define the feedback position in the shift register. The 0 and 1 feedback in one real Gold code generator is always the same (0 and 1 are the 2 M-code generators within one real Gold code generator). The TG..P. inputs are used to initialise the shift registers at reset or restart. The TG..G. inputs are used to program the polynomials to generate the Gold sequences.

The TG.CR\_JMP signal is used to generate a small section of the complete Gold code or to make a jump in the M-code when a certain state is reached. This is done in the following way :

For a restart : if LSFR 0 reaches the TG..S0 state, LSFR 0 and 1 are re-initialised with the TG..P. values in the next clock-cycle. TBC if this is sufficient, eg it will be very difficult/impossible to generate a code of length  $2^{41}$  as we have to predict the last state of that code. Otherwise we would need a 42 bit counter in stead of the rest compare registers.

For a jump : if LSFR 0 reaches the TG..S0 state, the INVERSE feedback value is shifted into LFSR 0 at the next clockcycle. So no jump is done in LFSR 1.

TG.CR_JMP	
00	none (complete Gold code)
01	restart (Gold or M-code)
10	jump (M-code)
11	undefined

There are no provisions to stop the generator for a number of chips or to jump forward a number of chips. This is related to the phasesteps in the spreaders.

The transfer of all Gold code parameters are synchronous with the symbol clock 0IA or 0IB. The selection between these 2 is done with the aid of the TG.CR\_RSCR parameter. Note that synchronous with a symbol clock means that the used datapath parameters can only change at the edge of a symbol. So during one symbol always the same parameter is used.

Note that TG.CR\_RSCR is also symbol clock synchronous with the symbol clock selected with TG.CR\_RSCR.

TG.CR_RSCR	parameter synchr. signal for Gold code generator .
0	symb clk 0IA
1	symb clk 0IB

66220-9245409

## 5.2 QPN channels with synchronisation hardware and PNcode generators

The transmitter contains 10 QPN channels. These 10 channels are combined in 2 sets of 4 QPN channels (set A and set B) and set C with 2 QPN channels.

Each set has a separate block for generating the PNcode and separate synchronisation hardware which defines the start of symbol transmission.

### 5.2.1 Synchronisation hardware with chip resolution

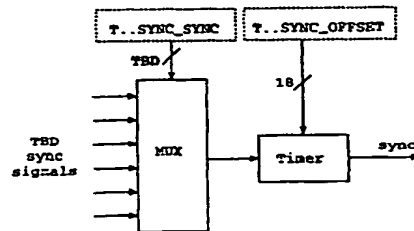


Figure 6: Channel set synchronisation hardware

The output of synchronisation hardware blocks A1, B1 and C go to the QPN channels of a set and define a common symbol start moment for all QPN channels in a set. This symbol start moment has a resolution of 1 chip. This signal is generated as a selection of 1 out of TBD(#) TBD incoming signals with a programmable offset. The T..SYNC\_SYNC signal is used as selector. (eg TA0SYNC\_SYNC). The incoming sync channels can be generated by eg : external pin (another chip) with synchroniser, RX timers (symbclk, slot edge, frame edge, superframe edge), TX timers (symbclk, slot edge, frame edge, superframe edge), acq hardware (FAU output edge). All TBD , first UMTS review needed !!!

The external pin is called TSSYNC and is synchronized at chiprate.

To generate the offset a counter at  $f_c$  rate can be used. This gives an offset resolution of 1 chip. The range of the offset is [0:262143]. This is sufficient to give an offset of 1 frame for UMTS in the 16.384 Mcps case.

2 extra Synchronisation hardware blocks A0 and B0 are foreseen to use as sync signals for the scrambling code generators, see later.

The transfer of the parameters is chip synchronous TBD.

## 5.2.2 QPN channel sets

There are 3 QPN channel sets as in fig 7.

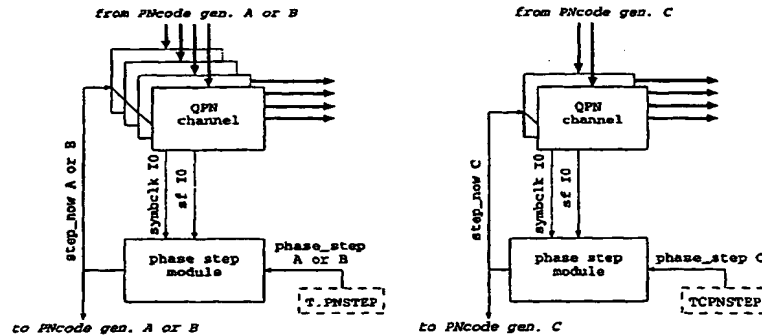


Figure 7: QPN channel set

Set A and B are identical and contain 4 QPN channels, set C contains only 2 QPN channels.

Each QPN channel has the following functional structure (fig 8).

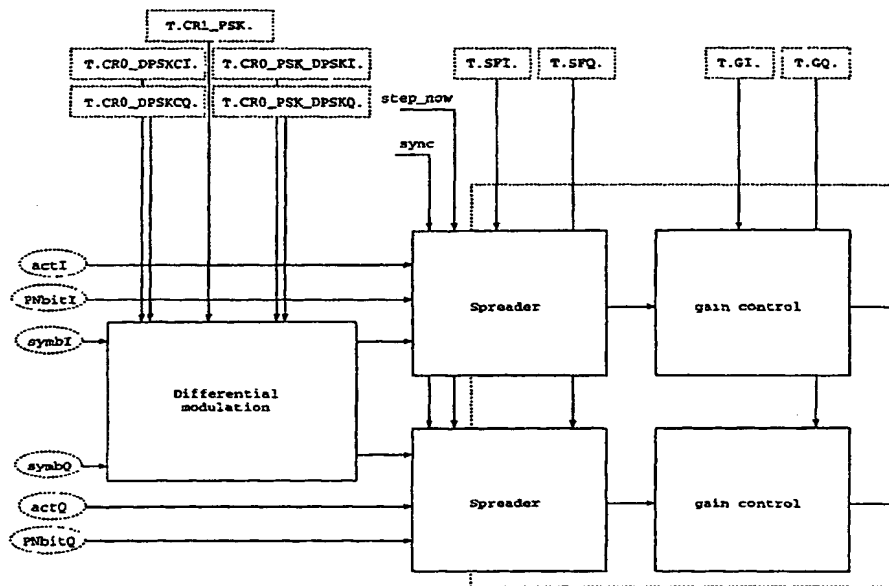


Figure 8: QPN channel structure

## 5 TRANSMITTER SPECIFICATION

## Differential modulation module

This module performs a DBPSK or DQPSK modulation. See fig 9.

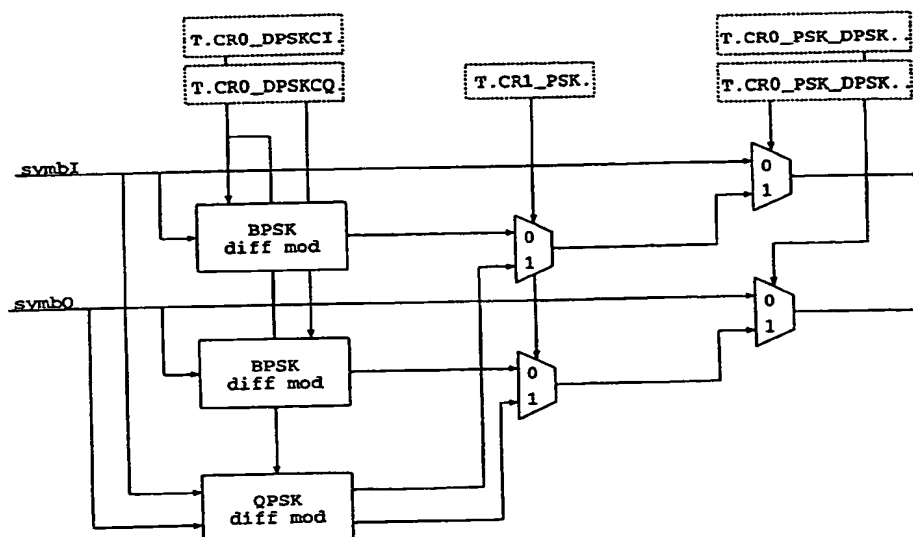


Figure 9: Differential modulation block

The function to perform depends on the T.CR0\_DPSKC.., T.CR0\_PSK\_DPSK.., T.CR1\_PSK. parameters. (eg TACR0\_DPSKCI0, TACR0\_PSK\_DPSKCI0, TACR\_PSK0 ). All parameters are 1 bit signals. T.CR1\_PSK. determines the PSK modulation BPSK or QPSK, T.CR0\_DPSKC.. the mode to use in the differential modulation scheme, T.CR0\_PSK\_DPSK.. determines if differential modulation is used for the branch.

Function of the 'BPSK diff mod' blocks :

T.CR0_DPSKC..	in(k)	out(k)	remarks
0	X	$\text{in}(k) \text{ XOR } \text{out}(k-1)$	DBPSK mode 0
1	X	$\text{NOT}(\text{in}(k) \text{ XOR } \text{out}(k-1))$	DBPSK mode 1

Function of the 'QPSK diff mod' block :

T.CR0_DPSKCI.	(inI(k), inQ(k))	(outI(k), outQ(k))	remarks
0	(0,0)	(outI(k-1), outQ(k-1))	DQPSK mode 0
	(0,1)	(outQ(k-1), NOT outI(k-1))	
	(1,0)	(NOT outQ(k-1), outI(k-1))	
	(1,1)	(NOT outI(k-1), NOT outQ(k-1))	
1	(0,0)	(outI(k-1), outQ(k-1))	DQPSK mode 1
	(0,1)	(NOT outQ(k-1), outI(k-1))	
	(1,0)	(outQ(k-1), NOT outI(k-1))	
	(1,1)	(NOT outI(k-1), NOT outQ(k-1))	

The 'BPSK diff mod' blocks have each a separate T.CR0\_DPSKC.. parameter (T.CR0\_DPSKCI. and

T.CR0\_DPSKCQ.), the 'QPSK diff mod' block is controlled by the T.CR0\_DPSKCI. parameter to choose the DPSK mode.

The differential module must take into account the activity bits of the different channels. The outI(k-1), outQ(k-1) signals are always symbols with an activity bit of 1.

Note that symbI and symbQ can be at different rates, the QPSK differential modes are only defined when both inputs are at the same rate.

With the T.CR0\_PSK\_DPSK.. parameter the differential modulation can be turned on/off for each branch.

T.CR0_PSK_DPSK..	differential modulation of branch ..
0	OFF
1	ON

This block has no functional delay, if for any reason there is a implementation delay, this delay should also be present when this block is not used.

The transfer of the parameters is synchronous with the symbol clock of the respective branch in the channel (eg TACR0\_PSK\_DPSKQ1 is synchronous with AQ1 symbol clock . T.CR1\_PSK. is synchronous with the I branch symbol clock of the channel.

#### Spreader and phasestepping

The input binary symbols, coming directly from the interface (symbI and symbQ) or coming from the Differential modulation module (depending on the T.CR\_PSK\_DPSK. parameter), are spread with the PNbits PNbitI and PNbitQ. Each symbol has an activity bit (actI and actQ). Functional spreader output :

symbol XOR PNbit	act	out (dec)
X	0	0
0	1	+1
1	1	-1

This activity bit can be used for burst transmission and for BPSK in stead of QPSK/QPN transmission. The implementation may be different by keeping the spreader output a 1 bit signal and taking into account the activity bit in the gain module.

The spreading factor is set by the T.SF.. inputs (eg TASFI0 is the spreading factor of the I branch of channel 0 of set A). The spreading factor (sf) is set to PNlength-1. So sf=0 is non-CDMA mode (no spreading).  $0 \leq sf \leq 32767$ . So the maximum spreading factor is 32768 (45 dB processing gain).

symbI and actI are signals at symbolrate  $fs_{xx}^I$ , symbQ and actQ are signals at symbolrate  $fs_{xx}^Q$ .  $fs_{xx}^I$  can differ from  $fs_{xx}^Q$ .

$$fc = fs_{xx}^I * (T.SFI. + 1) = fs_{xx}^Q * (T.SFQ. + 1).$$

The transfer of the T.SF.. parameter is synchronous with the symbol clock of the respective branch.

The spreaders can be (re)started via the sync signal (common for all spreaders in a set). Symbol clock signals (symclkI and symclkQ) are generated as a symbol reference for other hardware that requires symbol synchronous

actions, like the gain controls. These signals must also be available outside the chip. The symbol clocks are 1 before the first chip of a symbol. See fig 10 for timing between sync, symbols and symbol clocks. Note the delay of 2 chips between a sync pulse and the first chip of a symbol.

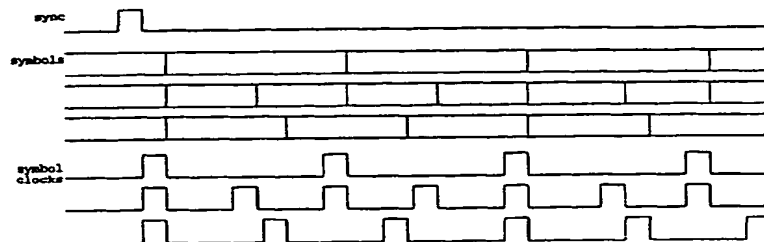


Figure 10: Spreaders timing

Per set a module is foreseen to do phasesteps. The phasesteps are done in the following way :

One of the 8 or 4 branches in a set is defined as reference branch (the I0 branch (TBC)), the symbol clock of this branch is used as a reference to do the phase steps. The resolution of the phasesteps is one chip, the range is [-32768:32767]. T.PNSTEP is a  $s < 16,0 >$  number. TBC. T.PNSTEP must be smaller than T.SFIO. There are some more limitations, see section 5.2.3.

A positive phasestep is an advance, a negative phasestep is a delay.

When a phasestep is done, some action should be done in the PNcode generators. The step\_now signal could also be used for that. See section 5.2.3 for more info.

The phasestep register (T.PNSTEP) is a self-clearing register. Ie when applying the phasestep the register is cleared.

The phasestep are performed :

This is only an explanation of what happens with the symbol clocks of the different branches, the PN codebits generation used during phasestepping is explained in section 5.2.3.

For a delay : at the next symbol edge of the reference branch in a set, the reference symbol clock is NOT produced, but is delayed with the number of chips we have to delay. All the other branches in the set do NOT produce symbol clocks in the 'delay period' and a symbol clock is generated for all the channels together with the reference symbol clock after the delay. So just after the 'delay period' all branches in the set are re-aligned. See fig 11

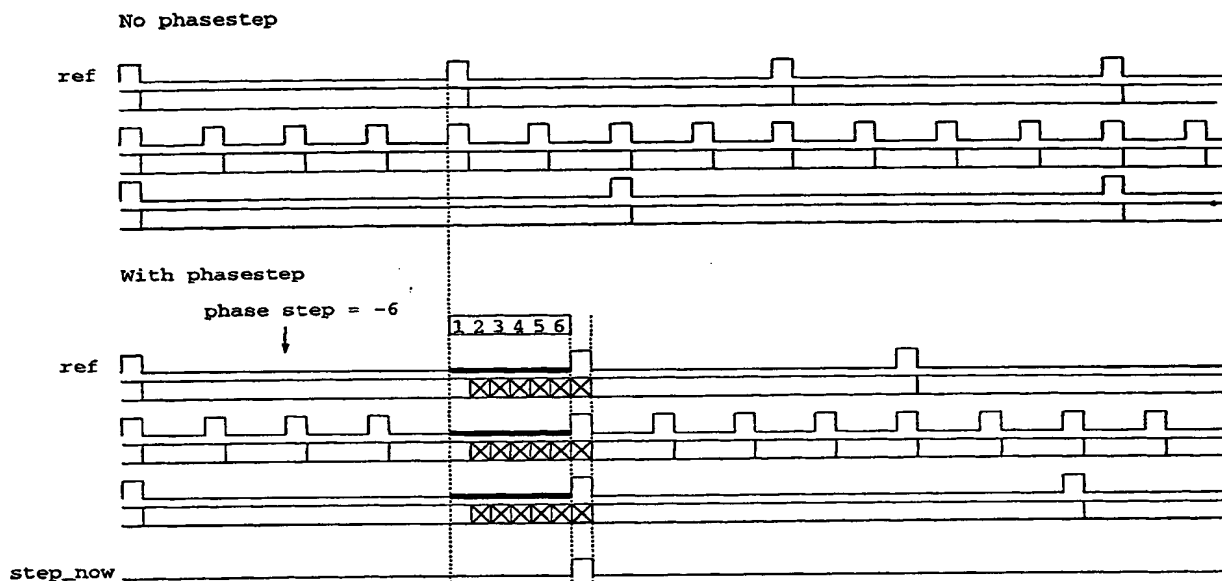


Figure 11: Phasestep - Delay



For an advance : at the next symbol edge of the reference branch in a set, nothing happens. The reference symbol edge AFTER it is advanced with the number of chips we have to advance. All the other branches in the set generate a symbol clock together with the advanced symbol clock of the reference branch. So at that time all branches in the set are re-aligned. See fig 12 Notice that it is possible that 2 symbol clocks come after each other.

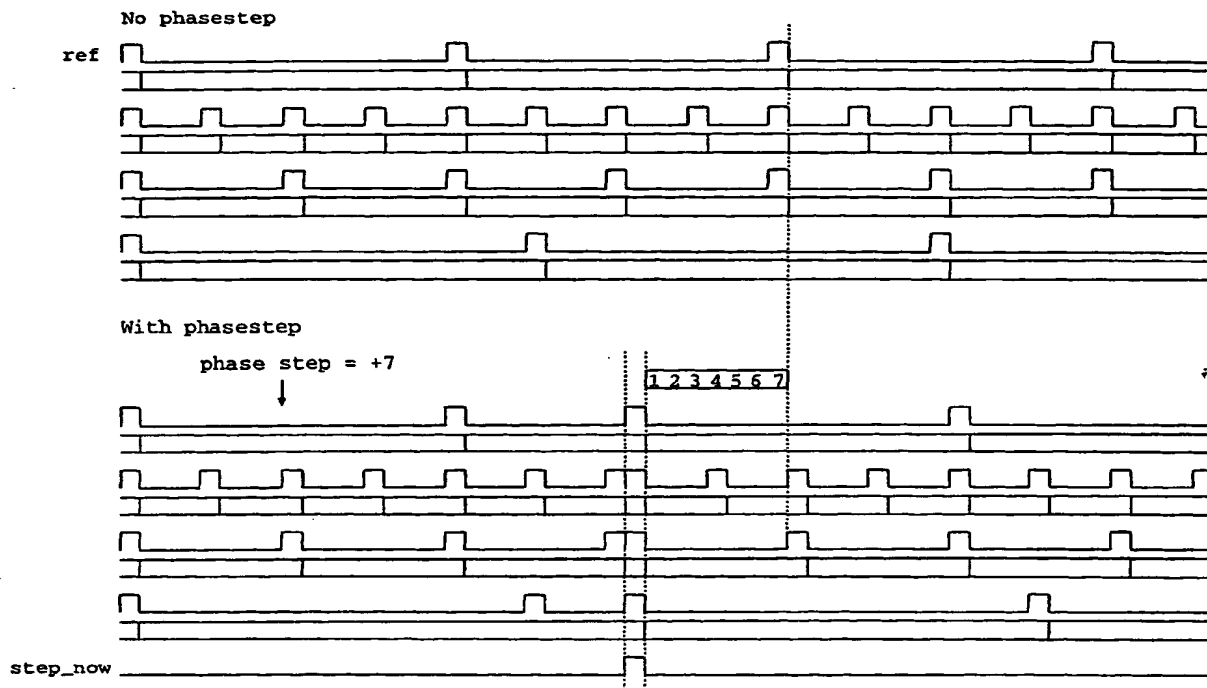


Figure 12: Phasestep - Advance

### Gain control

Each complex spreader is followed by a separate gain control. Each output branch of a spreader is again separately gain controlled.

20 dB range with a resolution of 0.2 dB was required. A  $u < 10, 6 >$  for the gains is used. In theory a  $u < 9, 6 >$  would be just enough to reach this with a 'strange' default value. i.e. if we use 000.101011B (=0.671875) as the smallest gain, we get a range of 21.5 dB with a minimum resolution of 0.2 dB. The usage of a  $u < 10, 6 >$  gives more room and flexibility. Eg if we use 0001.000000B (=1) as the smallest gain we get a range of 24 dB with a minimum resolution of 0.135 dB. So we could exchange range for resolution a bit more in this case.

The output of this block is a  $s < 11, 6 >$  number.

The gains must be applied symbol synchronous.

The update rate for each gain in UMTS is 6.4 kHz (16.384MHz/2560) TBC.  
Interrupts could be generated at this rate. See section 5.3

The transfer of the gains (T.G..) is synchronous with the respective symbol clocks of the branch.

## 5.2.3 PNcode generators

These blocks generate the complex PN codes for the 9 channels. A codegenerator is foreseen per set. The PNcode generators for set A and B generate each 4 complex codes, while the generator for set C generates 2 complex PNcodes. Fig 13 gives an overview of the codegenerators for set A, fig 14 for set B and fig 15 for set C.

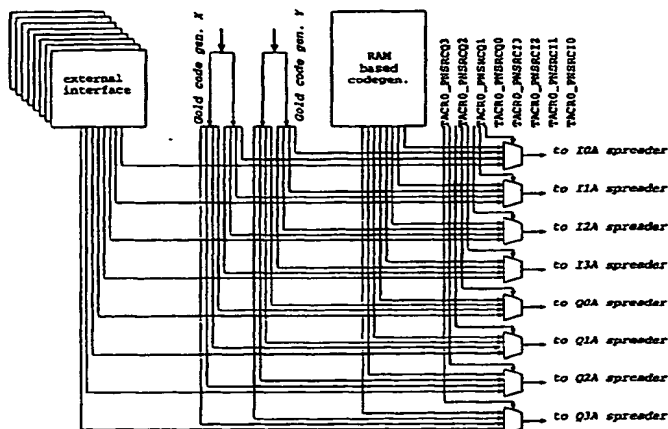


Figure 13: PNcode generator for set A

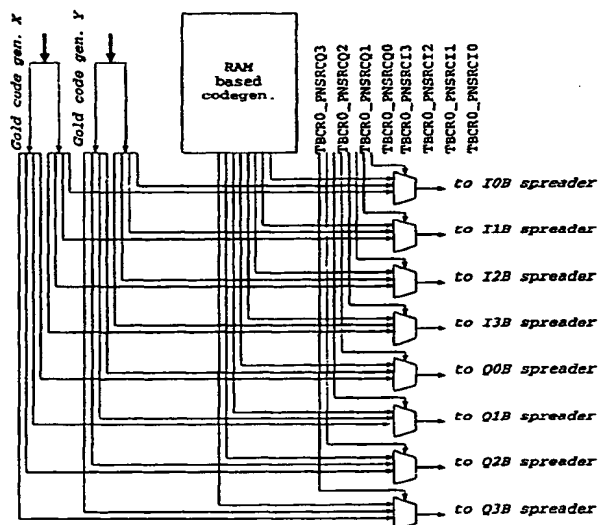


Figure 14: PNcode generator for set B

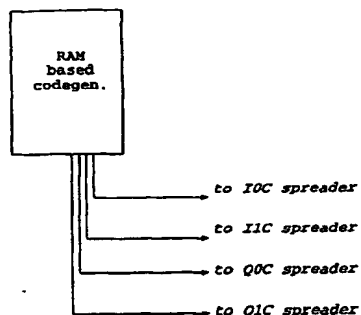


Figure 15: PNcode generator for set C

For set A three PNcode sources are possible : an external source, an input coming from a Gold code generator and a RAM based mechanism. Set B has no external PNcode source, set C is only RAM based.

The T.CR0.PNSRC.. inputs are used to select the correct source for the QPN channel branches. The T.CR0.PNSRC.. signals are 2 bit, the relation with the sources are :

T.CR0.PNSRC..	code source
00	Gold code generator X
01	Gold code generator Y
10	RAM
11	Extern

All PNbits are on  $f_c$  rate.

The transfer of the T.CR0.PNSRC.. parameters is synchronous with the respective branch of the respective channel.

#### External inputs for PNcode

Only for set A.

TBD interface.

TBD what to do when a phasestep is done in the spreaders.

#### Gold code generator input

The spreaders of set A and set B have the possibility to use one of the 2 Gold code generators as PNcode source. In this case, the Gold code generator output is directly used as spreading code.

As all channels are chip-synchronous, it is possible to use one Gold code generator for set A and set B at the same time. The other Gold code generator could then be used eg for a scrambling code generator.

The Gold code generators have no provisions to do phase-stepping.

## RAM based code generation

Each set has a block which can generate PNcodes based on a RAM. For all three sets the same block is used, except that for set C only the Q1,Q0,I1,I0 outputs are used. See fig 16.

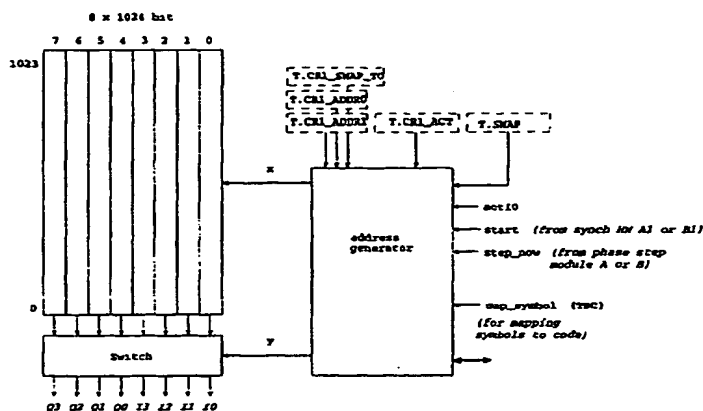


Figure 16: RAM based codegenerator

The block contains a RAM of  $8 \times 1024$  bits. An address generator selects one row of this RAM with the  $x$  address, then these 8 bit are routed to the spreaders via a switch controlled by address  $y$ . The switch contains 8 8to1 multiplexers and  $y$  is a  $8 \times 3 = 24$  bit address. (TBC)

The address generator has the following structure :

The signals to generate are programmed in a  $16 \times 48$  RAM = 768 bit RAM (16 TBC).

RAM $16 \times 48$			
	T.PNFSMX		
	JMP	XSTART	XEND
0	4bit	10bit	10bit
1			
2			
3			
14			
15			

Figure 17: address generator RAM

The RAM has 16 rows with in each row : T.PNFSMX\_JMP (4bit), T.PNFSMX\_XSTART (10 bit), T.PNFSMX\_XEND (10bit) and T.PNFSMY (24bit). When a new row in the RAM is accessed, x gets the value in T.PNFSMX\_XSTART of that row and x cnts down until equal to T.PNFSMX\_XEND. During that time y takes the T.PNFSMY value of the row. After x reached T.PNFSMX\_XEND : access RAM row defined in the T.PNFSMX\_JMP area.

The access to this RAM is controlled by a number of parameters and signals (functionally):

T.CR1\_ADDR0 and T.CR1\_ADDR1 are 2 pointers (4bit) to an address of the RAM. T.CR1\_SWAP\_TO is a pointer (1bit) to T.CR1\_ADDR0 and T.CR1\_ADDR1.

The start signal comes from synchronisation hardware blocks A1 or B1 and is used to (re)start the address generator. When  $start \cdot z^{-2}$  is 1: access ROW defined by the address defined in the address in T.CR1\_SWAP\_TO. The  $z^{-2}$  factor comes from the delay of 2 between sync pulse and the first chip of a symbol. See fig 18



Figure 18: time relation between sync, start and symbol edge.

The T.SWAP signal is used to swap between different codeblocks in the address generator RAM. When T.SWAP is 1 : access ROW defined by the address defined in the address in T.CR1\_SWAP\_TO. T.SWAP is symbol I0 synchronous and is only applied when the ARM has written a new 1 to it. T.SWAP is self cleared when applying.

This block could also be used for mapping a sequence of 4 symbols to a PNcode : just use the 4 bit symbol input to address the address generator RAM. This is not (yet) foreseen. TBD if needed.

It must be possible to hold the generator when the activity bit of channel 0, spreader I of the corresponding set is 0. So for the three sets always the activity bit I0 is taken as control for this. actI0 is only taken into account if T.CR1\_ACT is 1.

The parameters T.SWAP, T.CR1\_SWAP\_TO, T.CR1\_ADDR0, T.CR1\_ADDR1 and T.CR1\_ACT are transferred synchronous with the I0 branch of the set. T.SWAP is self clearing. The transfer of T.PNFSMX\_JMP, T.PNFSMX\_XSTART, T.PNFSMX\_XEND and T.PNFSMY is TBD.

Implementation requirement :

$T.PNFSMX\_XSTART - T.PNFSMX\_XEND \geq 1$ . (T.PNFSMX\_XSTART and T.PNFSMX\_XEND in same RAM row.)

When a phasestep is done, some action is needed for generating the right PNbits. The step\_now signal can be used for this. The step\_now signal is 1 together with the realigned symbol clocks at the end of the phasestep process (so before the first chip of realigned symbols). See fig 11 and 12.

For an advance :

The PNbits (of all the branches in the set) until and including step\_now equal to 1, are the same PNbits as when no phasestep is done. When  $step\_now * z^{-1}$  is 1 : access ROW defined by the address defined in the address in T.CR\_SWAP\_TO.

One extra implementation requirement :  $modul_{T.PNFSMX\_XSTART - T.PNFSMX\_XEND + 1}(T.PNFSMX\_XSTART - T.PNFSMX\_XEND - T.PNSTEP) \geq 1$  TBC

For a delay :

The PNbits until and including step\_now equal to 1, are the same PNbits as when no phasestep is done. When  $step\_now * z^{-1}$  is 1 : access ROW defined by the address defined in the address in T.CR\_SWAP\_TO.

There is one exception : when a delay of one chip is done ( $T.PNSTEP = -1$ ) the one extra PNbit is not the PNbit as if there is no phasestep but is a 0 (all branches in the set).

The following figures contain some example of the usage and storing of codes in the PN RAM.

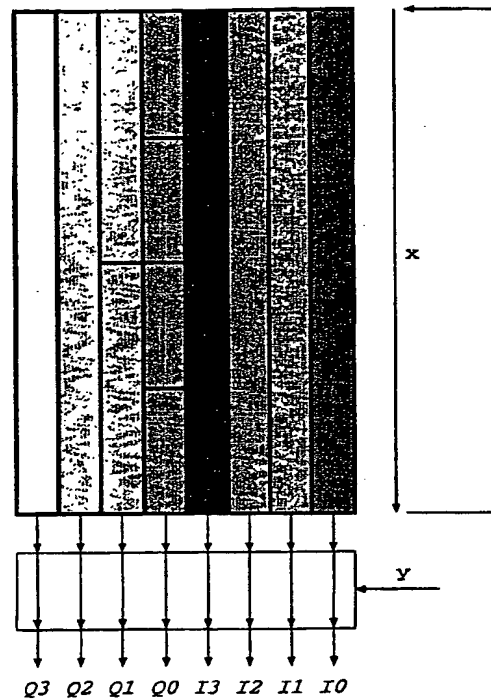


Figure 19: possible RAM configuration, example 1

Fig 19 : 8 BPSK (4 on I, 4 on Q) streams. Q3,Q2,I3,I2,I1,I0 have SF 1024, Q1 has SF 512 and Q0 has SF 256. x counts from 1023 to 0, y is a static TBD value.



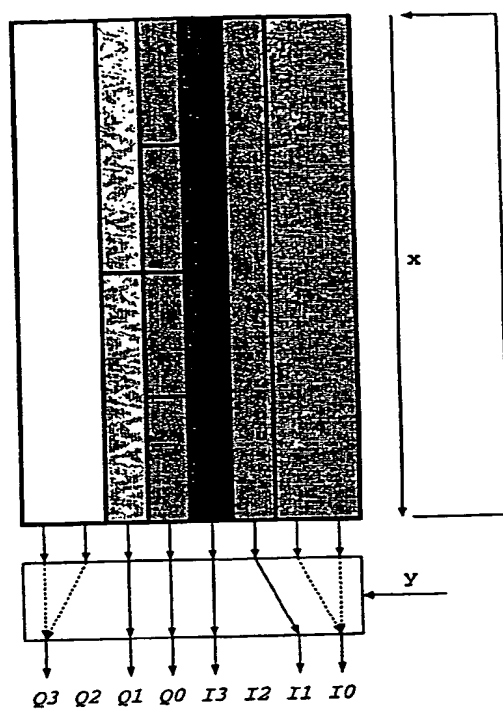


Figure 20: possible RAM configuration, example 2

Fig 20 : 6 BPSK (3 on I, 3 on Q) streams. Q3 and I0 have SF 2048, Q1 has SF 512, Q0 has SF 256, I3 and I1 have SF 1024. QPN channel 2 is unused.  
x counts from 1023 to 0, y changes between 2 TBD values every 1024 chips.

## 5 TRANSMITTER SPECIFICATION



Figure 21: possible RAM configuration, example 3

Fig 21 : 2 QPSK streams, channel 0 has SF 256 and uses continuously the same code. channel 1 uses a sequence of 16 different codes of length 256.

This scheme is usable for SCH transmission if the address counter is stopped when the activity bit is 0. x counts from 1023 to 0, y changes between 4 TBD values every 1024 chips.

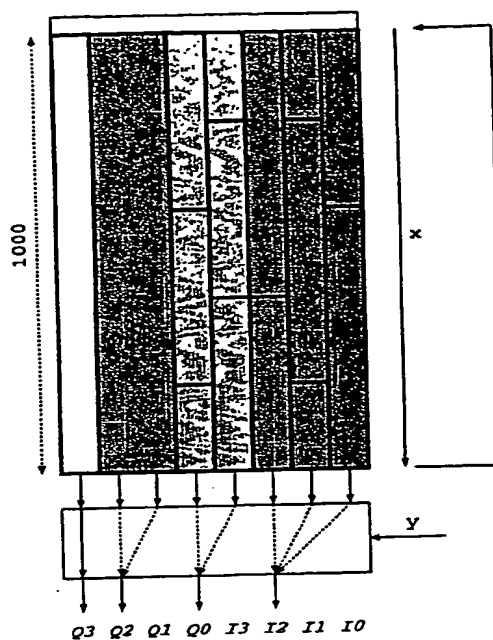


Figure 22: possible RAM configuration, example 4

Fig 22 : 4 BPSK streams, Q3 has SF 1000, Q2 has SF 2000, Q0 has SF 400, I2 has SF 600. Q1, I3, I1 and I0 spreaders are unused.  
x counts from 999 to 0, y changes between 3 TBD values every 1024 chips.

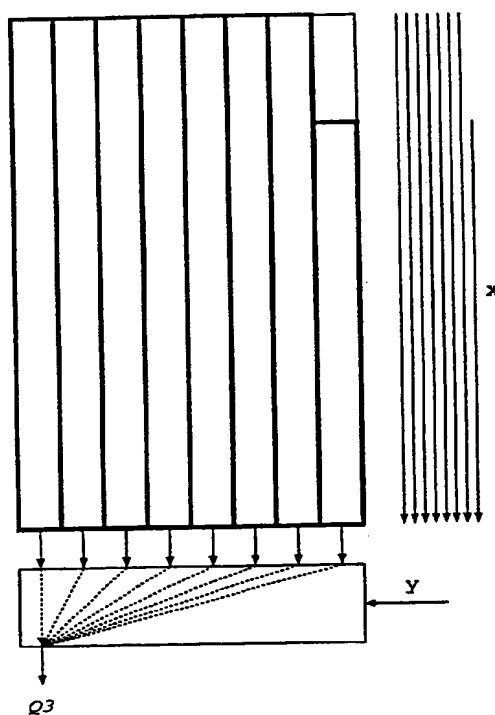


Figure 23: possible RAM configuration, example 5

Fig 23 : 1 BPSK stream on Q3 with SF 8000. x counts 7 times from 1023 to 0 and 1 time from 831 to 0. Every time x crosses 0, y changes between 8 TBD values.

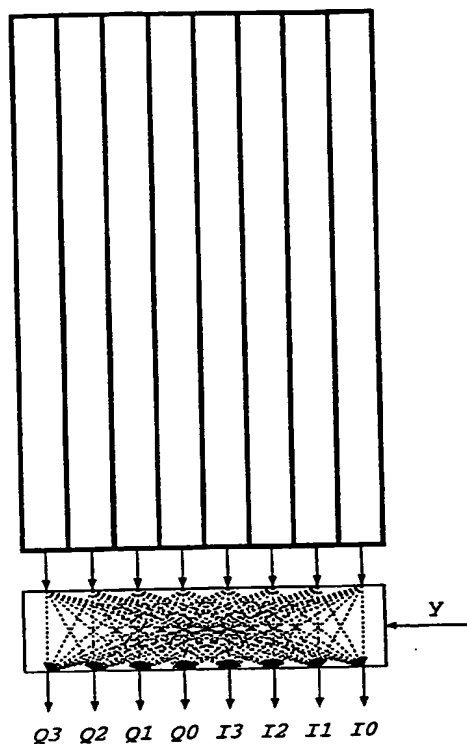


Figure 24: possible RAM configuration, example 6

Code shift modulation. 8 spreaders with code length 8192. The RAM stores 1 code of length 8192 with good autocorrelations.  $y$  changes between 8 TBD values.  $x$  counts from 1023 to 0.

As shown in figures 19, 20 and 22 in the case of variable spreading factor transmission it is assumed that spreading factors have a common multiple. The RAM is filled with replica's until the common multiple length is reached. In this way the symbols in one set are multiple-symbol synchronous.

It must be possible to write in the PNRAMs and FSMRAMs from the ARM without disrupting the normal reads at chip rate. See SL for more information. (extra address and data register in memory map, address of form 4000xx..., data transferred between chips when data written from ARM)

### 5.2.4 Synchronicity summary

This section gives an overview on synchronicity between, sets, channels, symbols and chips.  
See figure 25

	primary chip in set X	primary chip in set Y	RAM based symbol in set X	RAM based symbol in set Y	Extern based symbol (set A)	Gold based symbol in set A	Gold based symbol in set B	Transmission start channel in set X	Transmission start channel in set Y
primary chip in set X	S	S							
primary chip in set Y	S	S							
RAM based symbol in set X			MS	A	A	A	A		
RAM based symbol in set Y			A	MS	A	A	A		
Extern based symbol (set A)			A	A	A	A	A		
Gold based symbol in set A			A	A	A	S	A		
Gold based symbol in set B			A	A	A	A	S		
Transmission start channel in set X								S	A
Transmission start channel in set Y								A	S

Figure 25: Mutual synchronicity

S : completely synchronous at  $fc$  grid. Ie always the same start moment at  $fc$  grid.

A : completely asynchronous, the start moment of the two entries can have any offset of  $k/fc$  sec with  $k$  an integer value.

MS : multiple-symbol synchronous, this is defined at the end of the paragraph about RAM based code generators. This means that there must be a common multiple in symbol-length that fit in the RAM. Again this is at  $fc$  grid multiple-symbol synchronous.

### 5.3 Burst generator

This is a very TBD module which can be used to generate different signals like activity bits, restarts, .... for spreaders or codegenerators. In this way we can 'program' a waveform with bursts and 'other stuff' and we must only provide the data bits via an external interface but not repetitive, predictable things like an activity bit. Also frame and slot structures could be programmed here. Such a frame edge and slot edge (TBC) must be present as an external interrupt. This is also related with TDD mode.

< Must be studied in more detail, will contain a lot of counters at  $fc$  rate and some interrupt generators >.

### 5.4 Combiners A, B and C

The two combiners after set A and B at  $f_c$  rate, output the sum of the 4 incoming complex numbers. For combiner C this is the sum of 2 complex numbers.

There is no functional delay in this block, if for any reason latency is present in this block, the latency must be the same in every combiner.

The output of this block is a complex  $s < 13,6 >$  number.

### 5.5 Scrambler and scrambling code generator

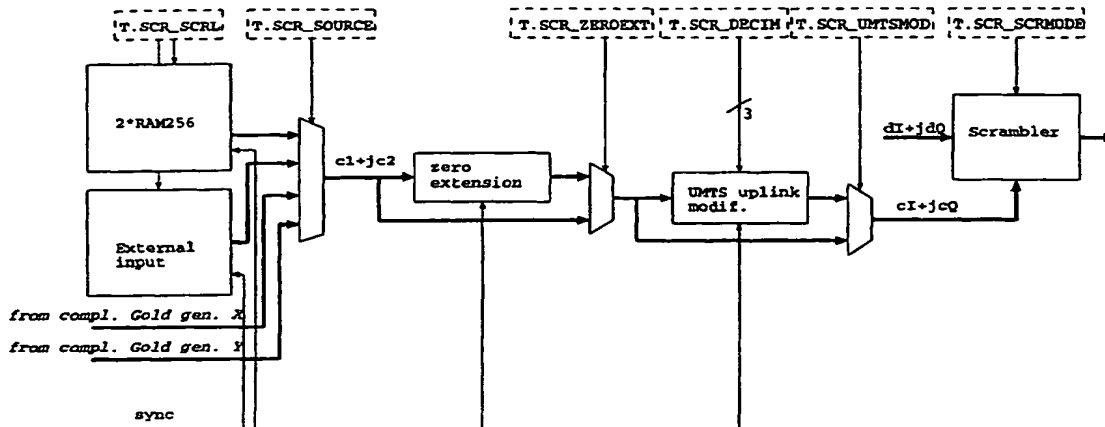


Figure 26: Scrambler with scrambling code generator

#### 5.5.1 Scrambling code generator

This block generates the complex scrambling code  $C_{scramb} = cI + jcQ$ .

It must be possible to start the generators synchronous with the spreaders, ie : sequence starts are aligned, taking hardware latency into account. So it must be possible to use the sync signal as a reference for the start of the scrambling code generator.

The generator can ONLY be (re)started via the sync signal. (TBC) Each scrambling code generator has its own Synchronisation hardware block to generate the sync signal. See fig 2 Note that the Gold code generator which output is an input to this block can also be restarted every IOA or IOB symbol.

Fig 27 gives the timing relation between sync pulse and scrambling code start.

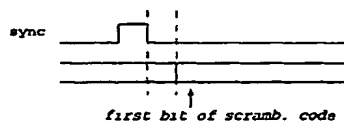


Figure 27: Scrambling code generator start

The delay of 2 chips between sync pulse and first bit of the scrambling code is needed because the start of the symbols have also a delay of 2 chips between sync pulse and first chip of a symbol. See fig 10.

The scrambling code generator has inputs for the 2 complex Gold code generators, 2 RAMs of 256 bit and an interface for external input of codes. With the T.SCR\_SOURCE signal we can select one of these 4 inputs.

T.SCR_SOURCE	code source
00	Gold code generator X
01	Gold code generator Y
10	RAM
11	Extern

It must be possible to use only the first k bits in the RAM, with k smaller than 257. For this reason the T.SCR\_SCRL input is present. A cyclic count from address T.SCR\_SCRL to 0 is done. The scrambling code length is T.SCR\_SCRL+1 with T.SCR\_SCRL maximal 255.

The 'UMTS uplink modif' block can modify the  $c1+jc2$  stream in the following way :

$$C_{scramb} = c1+jc2Q = c1(w_0 + jc2'w_1)$$

where  $w_0$  and  $w_1$  are chip rate sequences defined as repetitions of:

$$w_0 = \{1 \ 1\}$$

$$w_1 = \{1 \ -1\}$$

and where  $c_1$  is a real chip rate code, and  $c_2'$  is a decimated version of the real chip rate code  $c_2$ . The preferred decimation factor is 2, however other decimation factors should be possible in future evolutions of UMTS if proven desirable.

With a decimation factor of 2,  $c_2'$  is given as :  $c_2'(2k)=c_2'(2k+1)=c_2(2k)$ ,  $k=0,1,2...$

The T.SCR\_DECIM input is used to set the decimation factor. The decimation factor is T.SCR\_DECIM+1. T.SCR\_DECIM is a  $u < 3, 0 >$  number, so the maximum decimation factor is 8. What happens with  $w_0$  and  $w_1$  if the decimation factor is larger than 2?

With T.SCR\_UMTSMOD the usage of the 'UMTS uplink modif' block

T.SCR_UMTSMOD	UMTS modif ?
0	NO
1	YES

The 'zero extension' block can modify the  $c1+jc2$  stream in the following way :

$c1+jc2Q$  is  $c1+jc2$  with a zero in front.  $c1 = < 0, c1 >$ ,  $c2Q = < 0, c2 >$ . Restart of the zero extension is only possible with a new sync pulse.

T.SCR_ZEROEXT	zero extension ?
0	NO
1	YES

There are no provisions to make a phasestep.

All parameters of scrambling code generator A are transfered synchronous with the IOA symbol clock.  
All parameters of scrambling code generator B are transfered synchronous with the IOB symbol clock.



## 5 TRANSMITTER SPECIFICATION

## 5.5.2 Scrambler

The scrambling is in fact an overlay spreading without changing the chiprate.

Input data :  $dI + jdQ$  ( $s < 13, 6 >$ ).

Input scrambling code :  $cI + jcQ$  (+1 or -1).

This block has 3 modes :

T.SCR_SCRMODE	output	remarks
00	input	Scrambling Off
01	$(dI + jdQ) * (cI + jcQ) = dI * cI - dQ * cQ + j(dI * cQ + dQ * cI)$	Complex scrambling
10	$dI * cI + j dQ * cQ$	Dual real scrambling

In the 3 modes the delay between in and output should be the same. Functionally there is no delay between input and output. If for any reason there is a delay between input and output, this delay must also be put at the end of set C.

With the complex scrambling 1 extra MSB is needed, the output of this block is thus a  $s < 14, 6 >$ .  $dI$  and  $dQ$  may not be the most negative values in a  $s < 13, 6 >$ , otherwise overflow is possible.

The TASC\_SCRMODE parameter is transferred synchronous with the IOA symbol clock.

The TBSCR\_SCRMODE parameter is transferred synchronous with the IOB symbol clock.

60145426-02209

## 5 TRANSMITTER SPECIFICATION

## 5.6 Combiner 3

This combiner outputs the sum of the 3 streams coming from the 2 scramblers and the stand-alone QPN channel. the output of this block was/is specified as a  $s < 15, 6 >$  with 8 QPN channels. If we keep this value with 10 QPN channels overflow is possible at this point!!!!!! Another possibility is to define the complex scrambling as  $(dI+jdQ)*(cI+jcQ)/2$  which would eliminate the overflow risk.

## 5.7 Interpolator with chip phase control

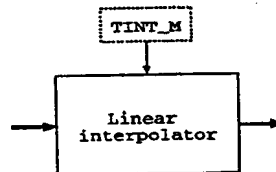


Figure 28: Interpolator with chip phase control

This block is used to do a chip phase shift with a resolution  $\ll 1$  chip. Only possibility to do a chip phase correction, not a chip frequency correction. So for every sample in, one output sample is generated, input and output clock is the equidistant  $f_c$  clock.

A linear interpolation is used to perform this function :

$$\text{out}(k) = (1-\text{TINT\_M}) * \text{in}(k-1) + \text{TINT\_M} * \text{in}(k)$$

where  $\text{in}(k-1)$  and  $\text{in}(k)$  are 2 consecutive equidistant complex samples at  $f_c$  rate. TINT\_M is an input of the block and is an  $u < 8, 8 >$  number ( $0 \leq \text{TINT\_M} < 1$ ).

$\text{in}(k)$  are  $s < 15, 6 >$  numbers, calculations are done full precision, the results are reduced (truncation of LSBs) to  $s < 15, 6 >$  numbers. So the input and output wordlengths are the same. No extra MSB is needed as the output of the linear interpolation can never be larger than the largest value of  $\text{in}(k)$ .

With  $\text{TINT\_M}=0$ ,  $\text{out} = \text{in} * z^{-1}$  (functional delay of 1 chip).

The TINT\_M parameter is transferred synchronous with the IOA symbol clock.

## 5.8 Upsampling4 and programmable filter

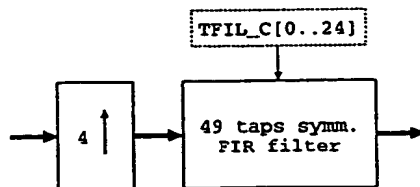


Figure 29: Upsampling4 and programmable filter

The fixed upsampling with a factor 4 (zero insertion) and a symmetrical programmable 49 taps filter are realized as a complex oversampling polyphase filter. The output sampling rate  $f_{4c} = 4 * f_c$ . The following table gives an overview of the wordlengths of the 25 programmable filter coefficients TFIL\_C[0..24].

Coeff. nr.	wordlength	max(abs(coef))
0 = 48	$s < 6, 12 >$	7.8125e-3
1 = 47	$s < 6, 12 >$	7.8125e-3
2 = 46	$s < 6, 12 >$	7.8125e-3
3 = 45	$s < 6, 12 >$	7.8125e-3
4 = 44	$s < 6, 12 >$	7.8125e-3
5 = 43	$s < 6, 12 >$	7.8125e-3
6 = 42	$s < 6, 12 >$	7.8125e-3
7 = 41	$s < 7, 12 >$	15.625e-3
8 = 40	$s < 7, 12 >$	15.625e-3
9 = 39	$s < 7, 12 >$	15.625e-3
10 = 38	$s < 8, 12 >$	31.25e-3
11 = 37	$s < 8, 12 >$	31.25e-3
12 = 36	$s < 8, 12 >$	31.25e-3
13 = 35	$s < 8, 12 >$	31.25e-3
14 = 34	$s < 9, 12 >$	62.5e-3
15 = 33	$s < 9, 12 >$	62.5e-3
16 = 32	$s < 9, 12 >$	62.5e-3
17 = 31	$s < 9, 12 >$	62.5e-3
18 = 30	$s < 10, 12 >$	125.0e-3
19 = 29	$s < 10, 12 >$	125.0e-3
20 = 28	$s < 10, 12 >$	125.0e-3
21 = 27	$u < 9, 12 >$	511/4096
22 = 26	$u < 10, 12 >$	1023/4096
23 = 25	$u < 11, 12 >$	2047/4096
24	$u < 11, 12 >$	2047/4096

The input data wordlength is a  $s < 15, 6 >$ . Inside the filter all multiplications and additions are done full precision and without overflow risk. The output of this is casted to a  $s < 21, 13 >$ . Dependent on the programmed coefficients overflow could be possible with this casting.

All filter coefficients TFIL\_C[0..24] are transferred synchronous with the IOA symbol clock.

## 5 TRANSMITTER SPECIFICATION

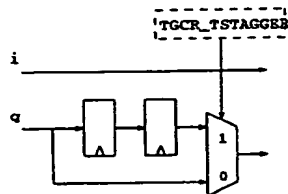
Example for a SRRC filter with roll-off 0.22 and sum of the 49 coefficients equal to 1.

Coeff. nr.	value
0 = 48	0.000732421875
1 = 47	0.002197265625
2 = 46	0.001220703125
3 = 45	-0.00146484375
4 = 44	-0.00341796875
5 = 43	-0.002685546875
6 = 42	0.001220703125
7 = 41	0.005615234375
8 = 40	0.006103515625
9 = 39	0.00097656250
10 = 38	-0.007568359375
11 = 37	-0.01318359375
12 = 36	-0.009521484375
13 = 35	0.003662109375
14 = 34	0.01904296875
15 = 33	0.02490234375
16 = 32	0.01220703125
17 = 31	-0.016357421875
18 = 30	-0.044677734375
19 = 29	-0.04980468750
20 = 28	-0.014404296875
21 = 27	0.061279296875
22 = 26	0.155517578125
23 = 25	0.23339843750
24	0.26367187500

The filter is used as a 4 time oversampling filter so only 1/4 of the 49 coefficients is used at the same time. The maximum gain we can have with the filter is 0.4585. So the  $s < 21, 13 >$  with 8 bits before the decimal point is enough with a  $s < 15, 6 >$  input (9 bits before the decimal point).

## 5.9 Offset modulation

By setting TGCRTSTAGGER to 1, the Q branch will be delayed with 0.5 chip. See fig 30.

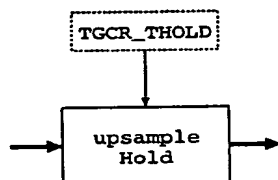
Figure 30: *offset modulation*

TGCRTSTAGGER	offset modulation ?
0	OFF
1	ON

TGCRTSTAGGER is transferred synchronous with the IOA symbol clock.

## 5.10 Hold 1-1024

This block performs a hold upsampling with a factor TGCRTHOLD+1.  $f_o = (TGCRTHOLD + 1) * f_{4c}$ . TGCRTHOLD can be any integer in the interval [0:1023].

Figure 31: *Upsampling-Hold*

TGCRTHOLD is transferred synchronous with the IOA symbol clock.

## 5.11 Complex upconverter and NCO

## 5.11.1 NCO

This block generates a cosine and sine value. The cos and sin values are frequency and phase controllable. See fig 33

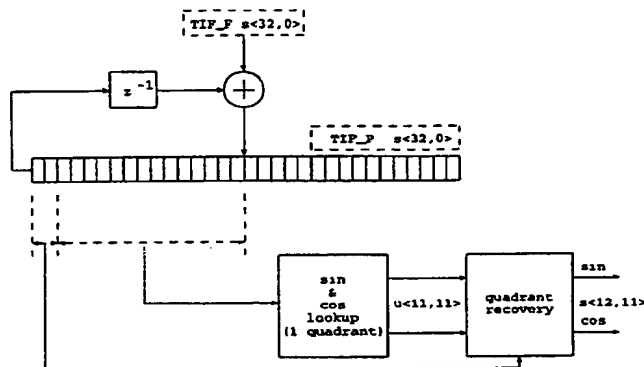


Figure 32: NCO with frequency and phase control

The TIF\_F and TIF\_P registers are both  $s < 32, 0 >$  registers in the memory map, controllable by the ARM.

The sine and cosine values are generated with the 16 MSB of a  $s < 32, 0 >$  TIF\_P value. The 14 LSB of this 16 bit number go to 2 lookup tables which contain the values for sin and cos in  $[0, \pi/2[$  with a gain of 2047/2048. The lookup wordlength for sin and cos in quadrant 1 is  $u < 11, 11 >$ . The 2 MSB of the  $s < 32, 0 >$  bit phase register are used to recover the quadrant, sin and cos are  $s < 12, 11 >$  numbers. The output of the NCO is the complex signal  $(\cos + j.\sin)$ . cos and sin are symmetrical signals containing all  $s < 12, 11 >$  values in the range  $[-2047/2048:2047/2048]$ .

The phase of the sin and cos can be controlled directly by writing to the TIF\_P register and setting TIF\_F to 0. For frequency control the TIF\_F value is integrated with wrap around to get the TIF\_P register. The TIF\_F can be used to program the frequency of the generated sine and cosine in the following way :

$$f_{sin} = f_{cos} = TIF\_F / 2^{32} * f_o.$$

With TIF\_F negative a negative (complex) IF will be generated.

Eg to generate a complex carrier at -20 MHz with  $f_o = 80MHz$ , TIF\_F should be set to -1073741824(dec) = C0000000(Hex).

The NCO parameters TIF\_F and TIF\_P are transferred synchronous with the IOA symbol clock. TIF\_P however is only transferred when the ARM has written a new value to the interface for it, otherwise TIF\_P is the wrap around integration of TIF\_F.

## 5 TRANSMITTER SPECIFICATION

### 5.11.2 Upconverter

Here a complex upconversion with the NCO generated complex carrier is done. See fig ??

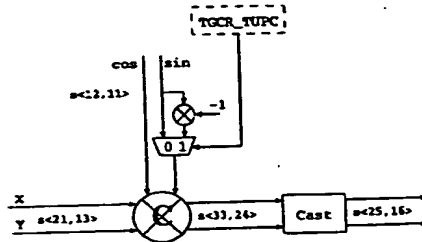


Figure 33: Upconverter

TQCR_TUPC	data in	carrier in	output
0	$X + jY$	$\cos + j\sin$	$(X + jY) * (\cos + j\sin)$
1	$X + jY$	$\cos + j\sin$	$(X + jY) * (\cos - j\sin)$

The computations are done full precision, the multiplications have 1 redundant bit as the most negative number will never be present in the sin or cos value. Thus the result of the multiplications are  $s < 32, 24 >$  bit numbers. This makes the full precision outputs  $s < 33, 24 >$  bit numbers. These full precision numbers are reduced to  $s < 25, 16 >$  numbers.

For information on phase noise, see SL.

TQCR\_TUPC is transferred synchronous with the IOA symbol clock.

## 5 TRANSMITTER SPECIFICATION

## 5.12 Level Control part 1

The purpose of the level control is to condition the signal coming from the upconverter prior to the 8 bit DA conversion or 12 bit digital output. This block is the first part of the level control, performing a coarse grain level control.

See fig 34.

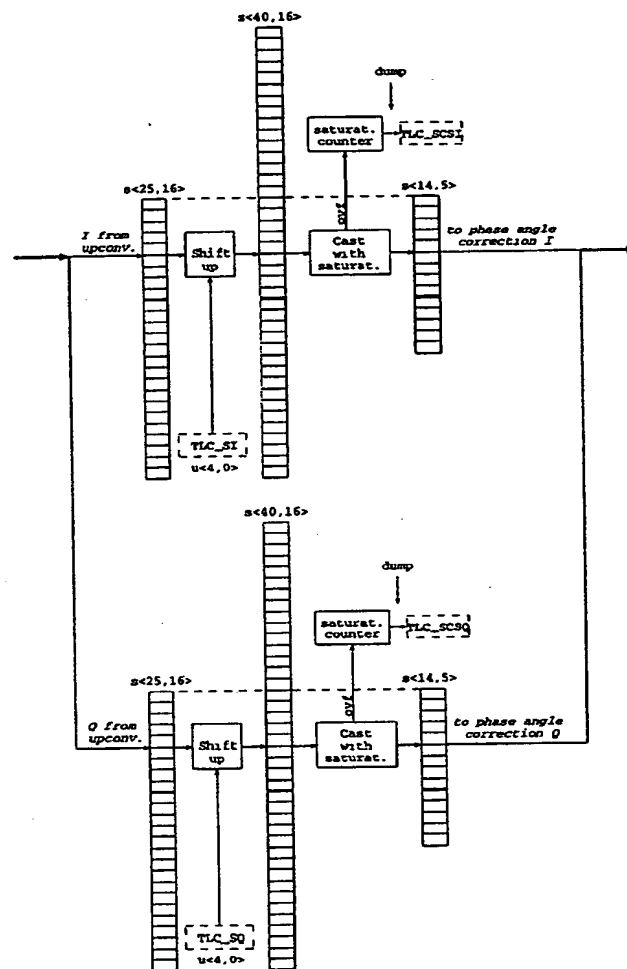


Figure 34: Level control part 1

The I and Q branch have a separate gain control, therefore there are separate gain factors for I and Q : TLC\_SI and TLC\_SQ.



## 5 TRANSMITTER SPECIFICATION

The incoming  $s < 25, 16 >$  signal is shifted up over TLC.S. (TLC.SI and TLC.SQ for i and q) bits towards the MSB with sign extension (so an operation equivalent with multiplying with  $2^{TLC.S}$ ). TLC.S. is a  $u < 4, 0 >$  number. The result is a  $s < 40, 16 >$  number. This  $s < 40, 16 >$  number is casted to a  $s < 14, 5 >$  with overflow detection and saturation (if the  $s < 40, 16 >$  number is larger than 000000000000000011111111.1111111111111111B (255.99998474121094D) or smaller than 1111111111111100000000.0000000000000000B (-256D) the  $s < 14, 5 >$  number is saturated to 01111111.111111B (255.96875D) or 10000000.000000B (-256D). The saturation counter counts the number of overflows/saturations in 4096 (TBD) samples. After 4096 (TBD) samples, the result is dumped in TLC.SCS. and the counter is restarted. For I and Q there are separate counters.

## 5.13 Phase angle compensation

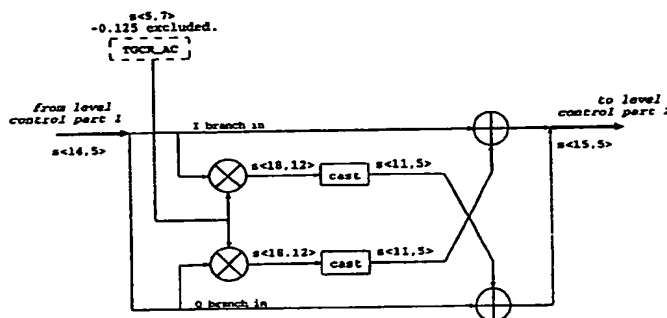


Figure 35: Phase angle compensation

A phase angel compensation is done :

$$I_{out} = I_{in} + Q_{in} * \tan(\beta)$$

$$Q_{out} = Q_{in} + I_{in} * \tan(\beta)$$

with  $2\beta$  the phase angle to correct.

Formulas are TBC !!!!

$\tan(\beta)$  is stored in the TGCRLAC variable. TGCRLAC is a  $s < 5, 7 >$  number with the most negative number (-0.125) excluded. This gives a range for  $2\beta$  larger than 13 degrees. The resolution of  $2\beta$  is smaller than 1 degree.

As the most negative number is not present in TGCRLAC, the result of the multiplication of  $I_{in}$  or  $Q_{in}$  with TGCRLAC is a  $s < 18, 12 >$  number. This is casted to a  $s < 11, 5 >$  number. The output wordlength is a  $s < 15, 5 >$  number.

When angle correction is off (TGCRLAC = 0), the output  $s < 15, 5 >$  number has 1 insignificant MSB. With maximum angle correction (TGCRLAC =  $\pm 0.1171875$ ) the extra MSB from input to output is only used for  $\approx 12\%$ .

The transfer of TGCRLAC is synchronous with the IOA symbol clock.

## 5.14 Level Control part 2

This is the high resolution of the level control. See fig 36.

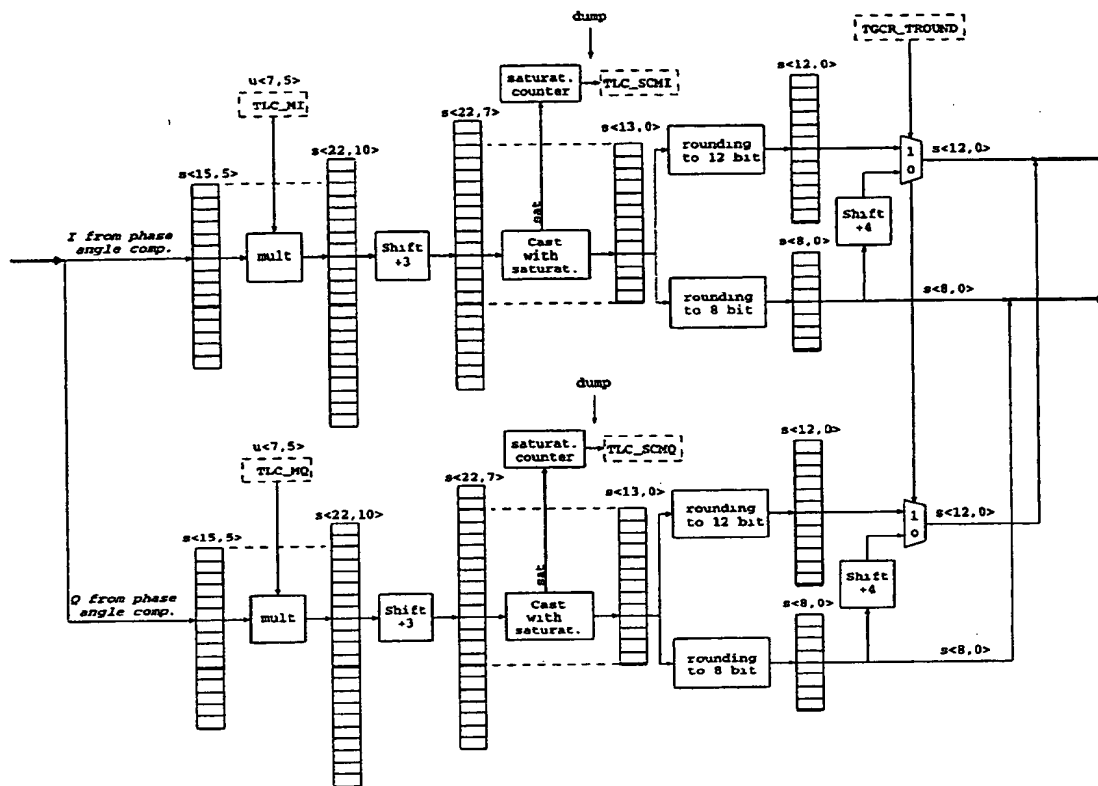


Figure 36: Level control part 2

The incoming  $s < 15,5 >$  number is multiplied with TLC.M. ( $u < 7,5 >$ ) (TLC.MI and TLC.MQ for i and q). The result is a  $s < 22,10 >$  number. This number is shifted up 3 bits (functionally only) which gives a  $s < 22,7 >$  number.

This  $s < 22,7 >$  number is casted to a  $s < 13,0 >$  number with overflow detection and saturation (if the  $s < 22,7 >$  number is larger than 0001111111111111111111B (4095.9921875D) or smaller than 1110000000000000.0000000B (-4096D) the  $s < 13,0 >$  number is saturated to 0111111111111111B (4095D) or 1000000000000000B (-4096D)). The saturation counter counts the number of saturations in 4096 (TBD) samples. After 4096 (TBD) samples, the result is dumped in TLC.SCM. (separate counter for I and Q) and the counter restarted.

When there is no phase angle compensation (TGCR.AC=0), TLC.M. can be up to 2 without possible saturation. With phase angle compensation TLC.M. must be between  $\approx 1.75$  and 2 in order not to have saturation risk. The actual value depends on the TGCR.AC parameter.

## 5 TRANSMITTER SPECIFICATION

This  $s < 13,0 >$  number is rounded to 8 and 12 bit. The method used for this rounding is the following : (eg for rounding to 8 bit) The signal is truncated to 9 bit, so with an extra LSB. When this 9 bit value is 01111111B the rounded 8bit value is 01111111B. When the 9 bit value is 10000000B the rounded 8 bit value is 10000001B. In all other cases the 8 bit rounded value is obtained by adding 00000001B to the 9 bit value and truncating to 8 bit (MSB). Eg when the 9 bit value is 110011001B this becomes 11001101B, when the 9 bit value is 110011000B this becomes 11001100B.

Depending on the TGCR\_TROUND input the  $s < 12,0 >$  number or the  $s < 8,0 >$  number (+4 '0' at the LSB side) goes to the  $s < 12,0 >$  output. The  $s < 8,0 >$  rounded number is the  $s < 8,0 >$  output of the level control.

TGCR_TROUND	pins output
0	rounded 8 bit + 4*'0'
1	rounded 12 bit

The transfer of the level control parameters is synchronous with the I0A symbol clock. TLC\_OC and TLC\_SC are read-only.

## 5.15 Transmitter output block

The transmitter has a complex 8 bit DAC output and a complex  $s < 12,0 >$  output to pins. The datapath values are multiplexed with registers in the memory map of the chip. See fig 37. Note that this is a figure for one branch (i or q) of the complex output.

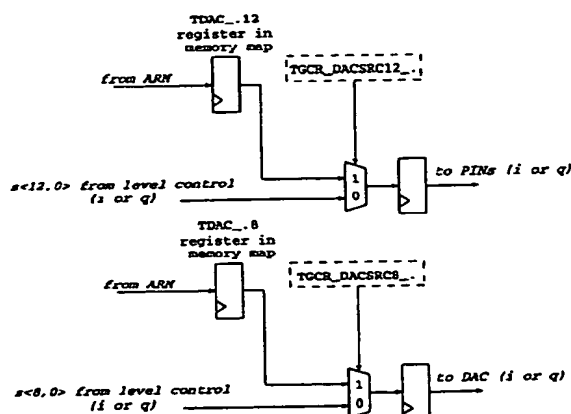


Figure 37: Transmitter output

The TGCR\_DACSRC12\_I, TGCR\_DACSRC12\_Q, TGCR\_DACSRC8\_I, TGCR\_DACSRC8\_Q select if the datapath or ARM outputs are used for the 4 transmitter outputs.

TGCR\_DACSRC12\_I, TGCR\_DACSRC12\_Q, TGCR\_DACSRC8\_I, TGCR\_DACSRC8\_Q are transferred synchronously with the symbol clock I0A. The inputs from the memory map (TDAC\_I12, TDAC\_Q12, TDAC\_I8, TDAC\_Q8) are not double buffered but go directly from the memory map to the MUX.

### 5.16 Asynchronous TX synchronisation

It is possible to re-initialise all TX time phasors via the external input pin TCSYNC. This pin is synchronised to the fastest ASIC clock and resets all timing phasors and its clock factory. Phasors are : symbol clk, chip clk, oversampled chip clocks. The resolution is 12.5 nsec.

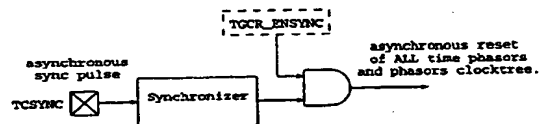


Figure 38: Asynchronous TX synchronisation signal

So after the asynchronous sync pulse on the finest grid of the chip all branches start with a new symbol. Other things like Gold/PN code generators are also restarted. (sync pulses A0, A1, B0, B1, C are 1 during async pulse)

During the synchronised asynchronous sync pulse the symbol clocks are forced to 0.

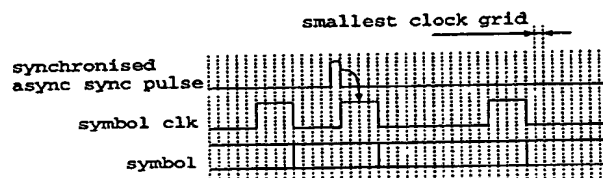


Figure 39: Asynchronous TX synchronisation effect

### 5.17 Global reset

The complete transmitter can be reset asynchronously by a reset pin. All parameters are set to a default value. And all channels behave as in fig 40.

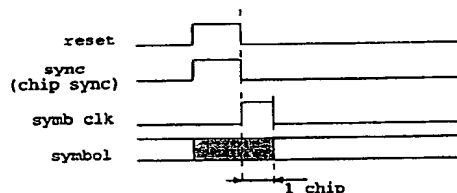


Figure 40: Asynchronous reset

During reset all chipsynchronous sync pulses (A0, A1, B0, B1, C) are 1. This has as effect that directly after reset goes to 0, another symbol clock is generated and 1 chip later the first chip of a new symbol starts. Other things like code generators are also restarted after reset has gone to 0 by the sync pulse which is 1 during reset.

During reset all symbol clocks are forced to 0.

## 5 TRANSMITTER SPECIFICATION

## 5.18 Transmitter parameters

This section gives an overview of all transmitter parameters in the memory map. The latency numbers are given in number of chips ( $= 1/f_c$  sec) and are measured from the generated 'parameter synchronisation signal'. When a parameter is synchronous with eg a symbol clock it means that the used datapath parameters can only change at the edge of a symbol. So during one symbol always the same parameter is used.

parameter	type	boot block	param. synch. signal	funct. latency	pipel. latency	remarks
TGXCR_RSRC	1bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXCR_SYNC	2bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXCR_SCLKRES	1bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXCR_MG	1bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXCR_JMP	2bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXCR_FEEDB	6bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXIG0	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXIG1	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXQG0	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXQG1	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXIP0	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXIP1	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXQP0	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXQP1	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXIS0	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGXQS0	42bit	Gold X	symblck I0A or I0B f(TGXCR_RSRC)	0	TBD	
TGYCR_RSRC	1bit	Gold Y	none/chip edge	0	TBD	
TGYCR_SYNC	2bit	Gold Y	symblck I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYCR_SCLKRES	1bit	Gold Y	symblck I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYCR_MG	1bit	Gold Y	symblck I0A or I0B f(TGYCR_RSRC)	0	TBD	

TGYCR_JMP	2bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYCR_FEEDB	6bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYIG0	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYIG1	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYQG0	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYQG1	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYIP0	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYIP1	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYQP0	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYQP1	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYIS0	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TGYQS0	42bit	Gold Y	symclk I0A or I0B f(TGYCR_RSRC)	0	TBD	
TA0SYNC_SYNC	TBD	set A	none/chip edge	0	TBD	
TA0SYNC_OFFSET	18bit	set A	none/chip edge	0	TBD	
TA1SYNC_SYNC	TBD	set A	none/chip edge	0	TBD	
TA1SYNC_OFFSET	18bit	set A	none/chip edge	0	TBD	
TB0SYNC_SYNC	TBD	set B	none/chip edge	0	TBD	
TB0SYNC_OFFSET	18bit	set B	none/chip edge	0	TBD	
TB1SYNC_SYNC	TBD	set B	none/chip edge	0	TBD	
TB1SYNC_OFFSET	18bit	set B	none/chip edge	0	TBD	
TCSYNC_SYNC	TBD	set C	none/chip edge	0	TBD	
TCSYNC_OFFSET	18bit	set C	none/chip edge	0	TBD	
TAPNSTEP	$s < 16, 0 >$	set A	transfer @ chip edge, self clearing on action	0 0	TBD	
TBPNSTEP	$s < 16, 0 >$	set B	transfer @ chip edge, self clearing on action	0 0	TBD	
TCPNSTEP	$s < 16, 0 >$	set C	transfer @ chip edge, self clearing on action	0 0	TBD	
TACR1_PSK0	1bit	set A	symclk I0A	0	TBD	
TACR1_PSK1	1bit	set A	symclk I1A	0	TBD	
TACR1_PSK2	1bit	set A	symclk I2A	0	TBD	
TACR1_PSK3	1bit	set A	symclk I3A	0	TBD	
TBCR1_PSK0	1bit	set B	symclk I0B	0	TBD	
TBCR1_PSK1	1bit	set B	symclk I1B	0	TBD	
TBCR1_PSK2	1bit	set B	symclk I2B	0	TBD	

TBCR1_PSK3	1bit	set B	symblck I3B	0	TBD
TCCR1_PSK0	1bit	set C	symblck I0C	0	TBD
TCCR1_PSK1	1bit	set C	symblck I1C	0	TBD
TACR0_DPSKC10	1bit	set A	symblck I0A	0	TBD
TACR0_DPSKCQ0	1bit	set A	symblck Q0A	0	TBD
TACR0_DPSKC11	1bit	set A	symblck I1A	0	TBD
TACR0_DPSKCQ1	1bit	set A	symblck Q1A	0	TBD
TACR0_DPSKC12	1bit	set A	symblck I2A	0	TBD
TACR0_DPSKCQ2	1bit	set A	symblck Q2A	0	TBD
TACR0_DPSKC13	1bit	set A	symblck I3A	0	TBD
TACR0_DPSKCQ3	1bit	set A	symblck Q3A	0	TBD
TBCR0_DPSKC10	1bit	set B	symblck I0B	0	TBD
TBCR0_DPSKCQ0	1bit	set B	symblck Q0B	0	TBD
TBCR0_DPSKC11	1bit	set B	symblck I1B	0	TBD
TBCR0_DPSKCQ1	1bit	set B	symblck Q1B	0	TBD
TBCR0_DPSKC12	1bit	set B	symblck I2B	0	TBD
TBCR0_DPSKCQ2	1bit	set B	symblck Q2B	0	TBD
TBCR0_DPSKC13	1bit	set B	symblck I3B	0	TBD
TBCR0_DPSKCQ3	1bit	set B	symblck Q3B	0	TBD
TCCR0_DPSKC10	1bit	set C	symblck I0C	0	TBD
TCCR0_DPSKCQ0	1bit	set C	symblck Q0C	0	TBD
TCCR0_DPSKC11	1bit	set C	symblck I1C	0	TBD
TCCR0_DPSKCQ1	1bit	set C	symblck Q1C	0	TBD
TACR0_PSK_DPSK10	1bit	set A	symblck I0A	0	TBD
TACR0_PSK_DPSKQ0	1bit	set A	symblck Q0A	0	TBD
TACR0_PSK_DPSK11	1bit	set A	symblck I1A	0	TBD
TACR0_PSK_DPSKQ1	1bit	set A	symblck Q1A	0	TBD
TACR0_PSK_DPSK12	1bit	set A	symblck I2A	0	TBD
TACR0_PSK_DPSKQ2	1bit	set A	symblck Q2A	0	TBD
TACR0_PSK_DPSK13	1bit	set A	symblck I3A	0	TBD
TACR0_PSK_DPSKQ3	1bit	set A	symblck Q3A	0	TBD
TBCR0_PSK_DPSK10	1bit	set B	symblck I0B	0	TBD
TBCR0_PSK_DPSKQ0	1bit	set B	symblck Q0B	0	TBD
TBCR0_PSK_DPSK11	1bit	set B	symblck I1B	0	TBD
TBCR0_PSK_DPSKQ1	1bit	set B	symblck Q1B	0	TBD
TBCR0_PSK_DPSK12	1bit	set B	symblck I2B	0	TBD
TBCR0_PSK_DPSKQ2	1bit	set B	symblck Q2B	0	TBD
TBCR0_PSK_DPSK13	1bit	set B	symblck I3B	0	TBD
TBCR0_PSK_DPSKQ3	1bit	set B	symblck Q3B	0	TBD
TCCR0_PSK_DPSK10	1bit	set C	symblck I0C	0	TBD
TCCR0_PSK_DPSKQ0	1bit	set C	symblck Q0C	0	TBD
TCCR0_PSK_DPSK11	1bit	set C	symblck I1C	0	TBD
TCCR0_PSK_DPSKQ1	1bit	set C	symblck Q1C	0	TBD
TASF10	$u < 15, 0 >$	set A	symblck I0A	0	TBD
TASF11	$u < 15, 0 >$	set A	symblck I1A	0	TBD
TASF12	$u < 15, 0 >$	set A	symblck I2A	0	TBD

TASFI3	$u < 15, 0 >$	set A	symblck I3A	0	TBD
TASFQ0	$u < 15, 0 >$	set A	symblck Q0A	0	TBD
TASFQ1	$u < 15, 0 >$	set A	symblck Q1A	0	TBD
TASFQ2	$u < 15, 0 >$	set A	symblck Q2A	0	TBD
TASFQ3	$u < 15, 0 >$	set A	symblck Q3A	0	TBD
TBSFI0	$u < 15, 0 >$	set B	symblck I0B	0	TBD
TBSFI1	$u < 15, 0 >$	set B	symblck I1B	0	TBD
TBSFI2	$u < 15, 0 >$	set B	symblck I2B	0	TBD
TBSFI3	$u < 15, 0 >$	set B	symblck I3B	0	TBD
TBSFQ0	$u < 15, 0 >$	set B	symblck Q0B	0	TBD
TBSFQ1	$u < 15, 0 >$	set B	symblck Q1B	0	TBD
TBSFQ2	$u < 15, 0 >$	set B	symblck Q2B	0	TBD
TBSFQ3	$u < 15, 0 >$	set B	symblck Q3B	0	TBD
TCSFI0	$u < 15, 0 >$	set C	symblck I0C	0	TBD
TCSFI1	$u < 15, 0 >$	set C	symblck I1C	0	TBD
TCSFQ0	$u < 15, 0 >$	set C	symblck Q0C	0	TBD
TCSFQ1	$u < 15, 0 >$	set C	symblck Q1C	0	TBD
TAGI0	$u < 10, 6 >$	set A	symblck I0A	0	TBD
TAGI1	$u < 10, 6 >$	set A	symblck I1A	0	TBD
TAGI2	$u < 10, 6 >$	set A	symblck I2A	0	TBD
TAGI3	$u < 10, 6 >$	set A	symblck I3A	0	TBD
TAGQ0	$u < 10, 6 >$	set A	symblck Q0A	0	TBD
TAGQ1	$u < 10, 6 >$	set A	symblck Q1A	0	TBD
TAGQ2	$u < 10, 6 >$	set A	symblck Q2A	0	TBD
TAGQ3	$u < 10, 6 >$	set A	symblck Q3A	0	TBD
TBGI0	$u < 10, 6 >$	set B	symblck I0B	0	TBD
TBGI1	$u < 10, 6 >$	set B	symblck I1B	0	TBD
TBGI2	$u < 10, 6 >$	set B	symblck I2B	0	TBD
TBGI3	$u < 10, 6 >$	set B	symblck I3B	0	TBD
TBGQ0	$u < 10, 6 >$	set B	symblck Q0B	0	TBD
TBGQ1	$u < 10, 6 >$	set B	symblck Q1B	0	TBD
TBGQ2	$u < 10, 6 >$	set B	symblck Q2B	0	TBD
TBGQ3	$u < 10, 6 >$	set B	symblck Q3B	0	TBD
TCGI0	$u < 10, 6 >$	set C	symblck I0C	0	TBD
TCGI1	$u < 10, 6 >$	set C	symblck I1C	0	TBD
TCGQ0	$u < 10, 6 >$	set C	symblck Q0C	0	TBD
TCGQ1	$u < 10, 6 >$	set C	symblck Q1C	0	TBD
TACR0_PNSRCI0	2bit	set A	symblck I0A	0	TBD
TACR0_PNSRCQ0	2bit	set A	symblck Q0A	0	TBD
TACR0_PNSRCI1	2bit	set A	symblck I1A	0	TBD
TACR0_PNSRCQ1	2bit	set A	symblck Q1A	0	TBD
TACR0_PNSRCI2	2bit	set A	symblck I2A	0	TBD
TACR0_PNSRCQ2	2bit	set A	symblck Q2A	0	TBD
TACR0_PNSRCI3	2bit	set A	symblck I3A	0	TBD
TACR0_PNSRCQ3	2bit	set A	symblck Q3A	0	TBD
TBCR0_PNSRCI0	2bit	set B	symblck I0B	0	TBD



## 5 TRANSMITTER SPECIFICATION

TBCR0_PNSRCQ0	2bit	set B	symblck Q0B	0	TBD
TBCR0_PNSRCI1	2bit	set B	symblck I1B	0	TBD
TBCR0_PNSRCQ1	2bit	set B	symblck Q1B	0	TBD
TBCR0_PNSRCI2	2bit	set B	symblck I2B	0	TBD
TBCR0_PNSRCQ2	2bit	set B	symblck Q2B	0	TBD
TBCR0_PNSRCI3	2bit	set B	symblck I3B	0	TBD
TBCR0_PNSRCQ3	2bit	set B	symblck Q3B	0	TBD
TACR0_SWAP_TO	1bit	set A	symblck I0A	0	TBD
TASWAP	1bit	set A	symblck I0A	0	TBD
			self clearing		
TACR1_ADDR0	4bit	set A	symblck I0A	0	TBD
TACR1_ADDR1	4bit	set A	symblck I0A	0	TBD
TACR1_ACT	1bit	set A	symblck I0A	0	TBD
TBCR1_SWAP_TO	1bit	set B	symblck I0B	0	TBD
TBSWAP	1bit	set B	symblck I0B	0	TBD
			self clearing		
TBCR1_ADDR0	4bit	set B	symblck I0B	0	TBD
TBCR1_ADDR1	4bit	set B	symblck I0B	0	TBD
TBCR1_ACT	1bit	set B	symblck I0B	0	TBD
TCCR1_SWAP_TO	1bit	set C	symblck I0C	0	TBD
TCSWAP	1bit	set C	symblck I0C	0	TBD
			self clearing		
TCCR1_ADDR0	4bit	set C	symblck I0C	0	TBD
TCCR1_ADDR1	4bit	set C	symblck I0C	0	TBD
TCCR1_ACT	1bit	set C	symblck I0C	0	TBD
TAPNFSMX_JMP[0..15]	4bit	set A	TBD	0	
TAPNFSMX_XSTART[0..15]	10bit	set A	TBD	0	
TAPNFSMX_XEND[0..15]	10bit	set A	TBD	0	
TAPNFSMY[0..15]	24bit	set A	TBD	0	
TBPNFSMX_JMP[0..15]	4bit	set B	TBD	0	
TBPNFSMX_XSTART[0..15]	10bit	set B	TBD	0	
TBPNFSMX_XEND[0..15]	10bit	set B	TBD	0	
TBPNFSMY[0..15]	24bit	set B	TBD	0	
TCPNFSMX_JMP[0..15]	4bit	set C	TBD	0	
TCPNFSMX_XSTART[0..15]	10bit	set C	TBD	0	
TCPNFSMX_XEND[0..15]	10bit	set C	TBD	0	
TCPNFSMY[0..15]	24bit	set C	TBD	0	
TASCR_SCLR	8bit	set A	symblck I0A	0	
TBSCR_SCLR	8bit	set B	symblck I0B	0	
TASCR_SOURCE	2bit	set A	symblck I0A	0	
TBSCR_SOURCE	2bit	set B	symblck I0B	0	
TASCR_ZEROEXT	1bit	set A	symblck I0A	0	
TBSCR_ZEROEXT	1bit	set B	symblck I0B	0	
TASCR_DECIM	$u < 3, 0 >$	set A	symblck I0A	0	
TBSCR_DECIM	$u < 3, 0 >$	set B	symblck I0B	0	
TASCR_UMTSMOD	1bit	set A	symblck I0A	0	

TBSCR_UMTSMOD	1bit	set B	symblck IOB	0		
TASCR_SCRMODE	2bit	set A	symblck IOA	0		
TBSCR_SCRMODE	2bit	set B	symblck IOB	0		
TINT_M	$u < 8, 8 >$	common	symblck IOA	0		
TFIL_C[0]	$s < 6, 12 >$	common	symblck IOA	1		
TFIL_C[1]	$s < 6, 12 >$	common	symblck IOA	1		
TFIL_C[2]	$s < 6, 12 >$	common	symblck IOA	1		
TFIL_C[3]	$s < 6, 12 >$	common	symblck IOA	1		
TFIL_C[4]	$s < 6, 12 >$	common	symblck IOA	1		
TFIL_C[5]	$s < 6, 12 >$	common	symblck IOA	1		
TFIL_C[6]	$s < 6, 12 >$	common	symblck IOA	1		
TFIL_C[7]	$s < 7, 12 >$	common	symblck IOA	1		
TFIL_C[8]	$s < 7, 12 >$	common	symblck IOA	1		
TFIL_C[9]	$s < 7, 12 >$	common	symblck IOA	1		
TFIL_C[10]	$s < 8, 12 >$	common	symblck IOA	1		
TFIL_C[11]	$s < 8, 12 >$	common	symblck IOA	1		
TFIL_C[12]	$s < 8, 12 >$	common	symblck IOA	1		
TFIL_C[13]	$s < 8, 12 >$	common	symblck IOA	1		
TFIL_C[14]	$s < 9, 12 >$	common	symblck IOA	1		
TFIL_C[15]	$s < 9, 12 >$	common	symblck IOA	1		
TFIL_C[16]	$s < 9, 12 >$	common	symblck IOA	1		
TFIL_C[17]	$s < 9, 12 >$	common	symblck IOA	1		
TFIL_C[18]	$s < 10, 12 >$	common	symblck IOA	1		
TFIL_C[19]	$s < 10, 12 >$	common	symblck IOA	1		
TFIL_C[20]	$s < 10, 12 >$	common	symblck IOA	1		
TFIL_C[21]	$u < 9, 12 >$	common	symblck IOA	1		
TFIL_C[22]	$u < 10, 12 >$	common	symblck IOA	1		
TFIL_C[23]	$u < 11, 12 >$	common	symblck IOA	1		
TFIL_C[24]	$u < 11, 12 >$	common	symblck IOA	1		
TGCR_TSTAGGER	1bit	common	symblck IOA	7		
TGCR_THOLD	$u < 10, 0 >$	common	symblck IOA	7		
TIF_F	$s < 32, 0 >$	common	symblck IOA	7		
TIF_P	$s < 32, 0 >$	common	symblck IOA	7		
			transf. only if ARM	7		
			wrote new value	7		
TGCR_TUPC	1bit	common	symblck IOA	7		
TGCR_TROUND	1bit	common	symblck IOA	7		
TLC_MI	$u < 6, 5 >$	common	symblck IOA	7		
TLC_MQ	$u < 6, 5 >$	common	symblck IOA	7		
TLC_SI	$u < 4, 0 >$	common	symblck IOA	7		
TLC_SQ	$u < 4, 0 >$	common	symblck IOA	7		
TLC_OC_I	$u < 12, 0 >$	common	N/A			read only
TLC_OC_Q	$u < 12, 0 >$	common	N/A			read only
TLC_SC_I	$u < 12, 0 >$	common	N/A			read only
TLC_SC_Q	$u < 12, 0 >$	common	N/A			read only
TGCR_DACSRC8_I	1bit	common	symblck IOA	7		

TGCR_DACSRC8.Q	1bit	common	symblck I0A	7		
TGCR_DACSRC12.Q	1bit	common	symblck I0A	7		
TGCR_DACSRC12.Q	1bit	common	symblck I0A	7		
TDAC_I8	$s < 8, 0 >$	common	no transfer reg.			
TDAC_Q8	$s < 8, 0 >$	common	no transfer reg.			
TDAC_I12	$s < 12, 0 >$	common	no transfer reg.			
TDAC_Q12	$s < 12, 0 >$	common	no transfer reg.			
TGCR_ENSYNC	1bit					

60445436.072399

## 6 Receiver Specification

The global receiver structure is shown in fig 41. All functional blocks are discussed in more detail in the next paragraphs.

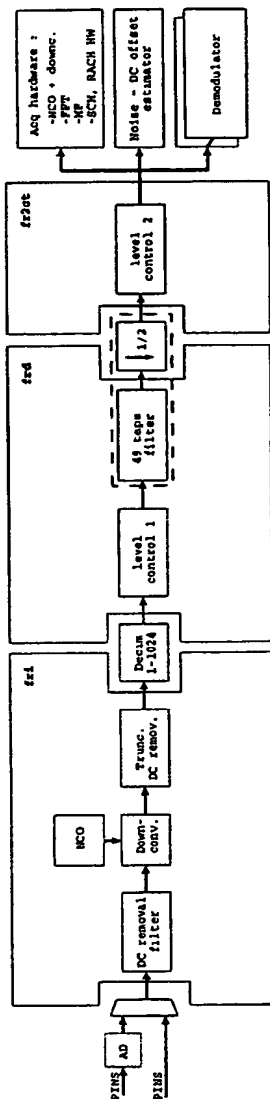


Figure 41: Global RX structure

## 6 RECEIVER SPECIFICATION

### 6.1 Input sampling and selection

The input of the digital part can come from an 8 bit AD or from external pins, also on 8 bit (TBC). The input of the digital part is at  $f_{ri}$  rate. When the input comes from the AD,  $f_{ri}$  is maximal 40 MHz, with the external inputs it is possible to have  $f_{ri}$  at 80 MHz.

The input is a  $s < 8, 0 >$  number.

### 6.2 DC removal filter

This is a TBD filter to remove a DC component in the input signal.  $s < 8, 0 >$  input. Implementation is TBD. Output wordlength is  $s < 10, 2 >$ . It must be possible to turn this filter off.

### 6.3 Common downconverter with NCO

#### 6.3.1 NCO

This block generates a cosine and sine value. The cos and sin values are frequency, phase and phase step controllable. See fig 42

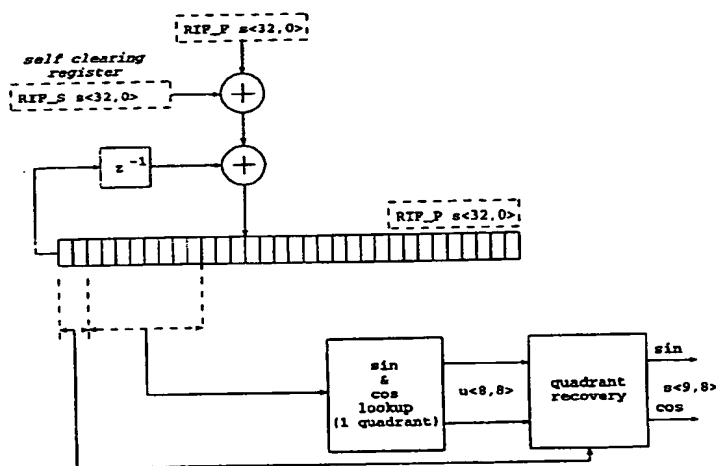


Figure 42: common NCO with frequency, phase and phasestep control

The sine and cosine values are generated with the 10 MSB of a  $s < 32, 0 >$  phase value. The 8 LSB of this 10 bit number go to 2 lookup tables which contain the (256) values for sin and cos in  $[0, \pi/2[$  with a gain of 255/256. The lookup wordlength for sin and cos in quadrant 1 is  $u < 8, 8 >$ . The 2 MSB of the  $s < 32, 0 >$  bit phase register are used to recover the quadrant, sin and cos are  $s < 9, 8 >$  numbers. The output of the NCO is the complex signal  $(\cos + j.\sin)$ . cos and sin are signals in  $[-255/256:255/256]$ .

The  $s < 32, 0 >$  bit phase register can be directly controlled by writing to the RIF\_P register and setting RIF\_F to 0 and setting RIF\_S to 0. For frequency control the RIF\_F value is integrated with wrap around to get the RIF\_P register. The RIF\_F can be used to program the frequency of the generated sine and cosine in the following way :

$$f_{sin} = f_{cos} = RIF\_F / 2^{32} * f_{in}.$$

With RIF\_F negative a negative (complex) IF will be generated. (TBC)

Eg to generate a complex carrier at -20 MHz with  $f_{ri} = 80\text{MHz}$ , RIF\_F should be set to -1073741824.

With RIF\_S a one-time phase step can be given. The register should be auto-cleared after applying it.

RIF\_P, RIF\_F and RIF\_S are TBD-synchronous. RIF\_P is only transferred when the ARM has written a new value to it, otherwise RIF\_P is the wrap around integration of RIF\_F (and RIF\_S).

### 6.3.2 Downconverter

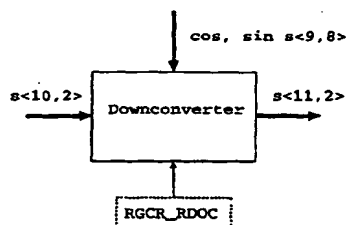


Figure 43: Common RX downconverter

This block performs a complex downconversion, with the NCO generated complex carrier, on the incoming complex signal. The output signal should be a near baseband signal.

RGCR_RDOC	data in	carrier in	output
00	$X + jY$	$\cos + jsin$	$(X + jY) * (\cos + jsin)$
01	$X + jY$	$\cos + jsin$	$(X + jY) * (\cos - jsin)$
10	$X + jY$	$\cos + jsin$	$2 * X * (\cos + jsin)$
11	$X + jY$	$\cos + jsin$	$2 * X * (\cos - jsin)$

The computations are done full precision, the multiplications have 1 redundant bit as the most negative number will never be present in the sin or cos value. Thus the result of the multiplications are  $s < 18, 10 >$  bit numbers. This makes the full precision outputs  $s < 19, 10 >$  bit numbers. These full precision numbers are reduced to  $s < 11, 2 >$  numbers.

The gain of 2 for the 1X modes is used to gain a significant LSB.

Input and output at  $f_{ri}$  rate.

### 6.4 Truncation DC removal

This block removes the truncation DC offset before going to the decimator. This is done by adding 1 LSB always equal to 1. So the outgoing wordlength is a  $s < 12, 3 >$  number. The last LSB of this number is always 1.

## 6.5 Decimation 1-1024

This block performs a decimation with a factor  $RGCR\_RDECIM+1$ . This is done by adding  $RGCR\_RDECIM+1$  incoming samples. The incoming samples are at  $f_{ri}$  rate, the output samples are at  $f_{rd}$  rate.

$$f_{rd} = f_{ri} / (RGCR\_RDECIM + 1).$$

$RGCR\_RDECIM$  can be any integer in the interval  $[0:1023]$ .

The decimator wordlength out is a  $s < 22, 3 >$  number.

*MAx-C-Mizer model :*

*the decimation phase is reset-controllable as in fig 44.*

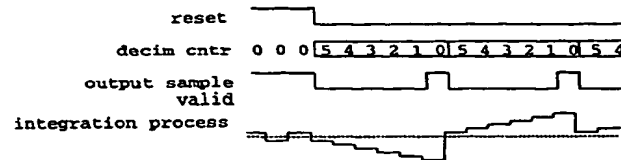


Figure 44: *MAx-C-Mizer decimation 1-1024 phase timing*

after reset the decimation phase restarts. This is an example with  $RGCR\_RDECIM$  equal to 5. A new output sample is available together with valid 1.

## 6.6 Level Control 1

The incoming  $s < 22, 3 >$  number is shifted down over  $R1LC\_S$  bits. This gives a  $s < 37, 18 >$  number. This number is shifted up 5 bits (No hardware, representation only). This number is casted with saturation to a  $s < 11, 2 >$  output number. Two complex saturation counters are present at this stage. Complex overflow counter 1 counts the real number of saturations, so the saturations where the saturation logic saturates the signal (if the  $s_{j37,13}_i$  is larger than 0000-000001111111.111111111111B (255.9998779296875D) or smaller than 1111-11111000000000.000000000000B (-256D) the  $s < 11, 2 >$  output is saturated to 01111111.11B (255.75D) or 10000000.00B (-256D) ). Complex overflow counter 2 counts the number of saturations as if the signal amplitude was twice as big (when the  $s_{j37,13}_i$  is larger than 0000-000011111111.111111111111B (511.9998779296875D) or smaller than 1111-11111000000000.000000000000B (-512D) ).

The result of the counters is dumped into  $R1LC\_SC1I$ ,  $R1LC\_SC1Q$ ,  $R1LC\_SC2I$  and  $R1LC\_SC2Q$  after 4096 input samples. At the same time the saturation counters are restarted.

With  $R1LC\_S$  equal to 15, the level control output is equal to the 11 MSB of the input signal. With  $R1LC\_S$  equal to 4, the level control output is equal to the 11 LSB of the input signal.

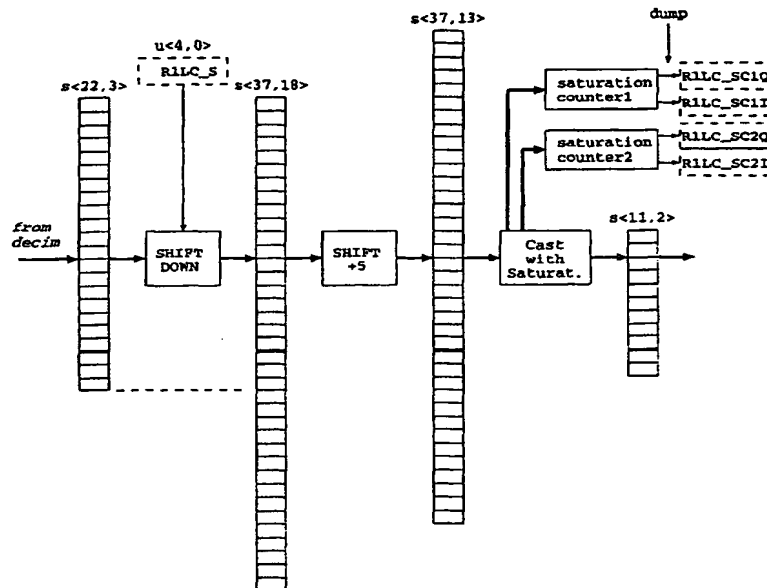


Figure 45: RX common level control



## 6.7 Programmable 49 taps FIR filter with programmable downsampling

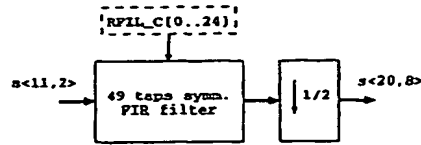


Figure 46: Programmable filter and downsampling

The complex receive stream coming from the downconverter is filtered by a programmable symmetrical 49 taps FIR filter and downsampled with a factor  $RX\_D+1$ .  $RX\_D$  can be 0 or 1.

Inputs are at  $frd$  rate, outputs at  $fr2ct$  rate.  $fr2ct = frd$  or  $fr2ct = frd/2$  depending on  $RX\_D$ .

The following table gives an overview of the wordlengths of the 25 programmable filter coefficients  $RFIL\_C[0..24]$ .

Coeff. nr.	wordlength	max(abs(coef))
0 = 48	$s < 6, 12 >$	7.8125e-3
1 = 47	$s < 6, 12 >$	7.8125e-3
2 = 46	$s < 6, 12 >$	7.8125e-3
3 = 45	$s < 6, 12 >$	7.8125e-3
4 = 44	$s < 6, 12 >$	7.8125e-3
5 = 43	$s < 6, 12 >$	7.8125e-3
6 = 42	$s < 6, 12 >$	7.8125e-3
7 = 41	$s < 7, 12 >$	15.625e-3
8 = 40	$s < 7, 12 >$	15.625e-3
9 = 39	$s < 7, 12 >$	15.625e-3
10 = 38	$s < 8, 12 >$	31.25e-3
11 = 37	$s < 8, 12 >$	31.25e-3
12 = 36	$s < 8, 12 >$	31.25e-3
13 = 35	$s < 8, 12 >$	31.25e-3
14 = 34	$s < 9, 12 >$	62.5e-3
15 = 33	$s < 9, 12 >$	62.5e-3
16 = 32	$s < 9, 12 >$	62.5e-3
17 = 31	$s < 9, 12 >$	62.5e-3
18 = 30	$s < 10, 12 >$	125.0e-3
19 = 29	$s < 10, 12 >$	125.0e-3
20 = 28	$s < 10, 12 >$	125.0e-3
21 = 27	$s < 11, 12 >$	0.25
22 = 26	$s < 11, 12 >$	0.25
23 = 25	$u < 11, 12 >$	2047/4096
24	$u < 12, 12 >$	4095/4096

The input data wordlength is  $s < 11, 2 >$ . All multiplications and additions are done full precision and without overflow risk. The sum of  $\max(\text{abs}(\text{coef}))$  is  $\approx 4.7$ . So the full precision filter output is a  $s < 11 + 3 + 12, 2 + 12 > = s < 26, 14 >$  number.

The output of this is casted to a  $s < 20, 8 >$  bit number. No overflow is possible with this casting.

< See /users/tugil/CDMAx/filter-sets for a number of filter sets >

All filter coefficients  $RFIL\_C[0..24]$  are transferred TBD-synchronous.

There is only 1 filter and 1 common downconverter, this implies that if incoming streams with large IF differences enter this filter, there is some TBD loss due to mismatch of the incoming spectral shape and the filter shape.

< Make table : Max: 200 kHz @ 20 MHz, 3 dB BW = 1% mismatch and energy loss >

**MAx-C-Mizer model :**

the subsampling phase is reset-controllable as in fig 47.

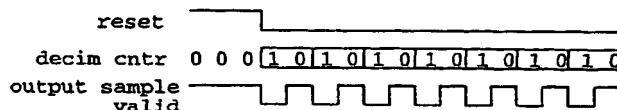


Figure 47: MAx-C-Mizer progr. 49 taps filter subsampling phase timing

after reset the decimation phase restarts. This is an example with  $RX\_D$  equal to 1. A new output sample is available together with valid 1.

## 6.8 Level control 2

To optimize the number of significant bits going into the demodulator correlators a common level control is foreseen to adapt the level of the signal coming from the filter. See 48 for the structure.

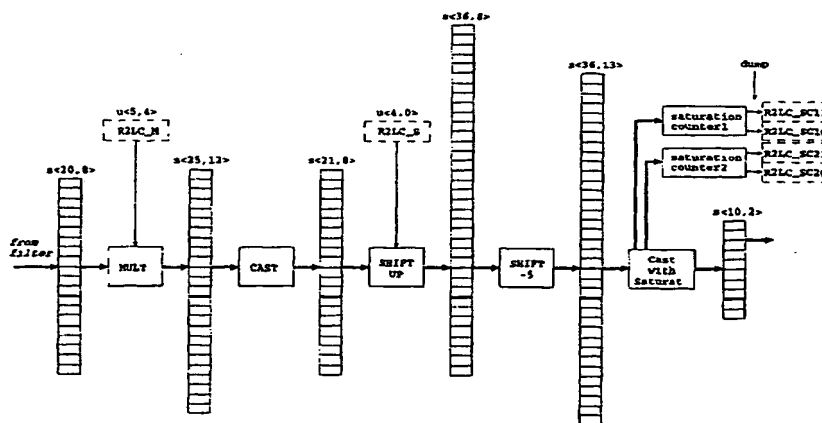


Figure 48: RX common level control

The incoming complex data  $s < 20, 8 >$  is multiplied with a  $u < 5, 4 >$  number. This results in a  $s < 25, 12 >$  number. So no overflow is possible here. This  $s < 25, 12 >$  number is casted to a  $s < 21, 8 >$  number. This  $s < 21, 8 >$  number is shifted up over R2LC\_S bits ([0:15]). This results in a  $s < 36, 8 >$  without overflow risk. This number is shifted down 5 bits into a  $s < 36, 13 >$  (no hardware cost, representation only). This result is casted with

saturation into the  $s < 10, 2 >$  output number ( $20-8+1-5=10-2$ ). Two complex saturation counters are present at this stage. Complex overflow counter 1 counts the real number of saturations, so the saturations where the saturation logic saturates the signal (if the  $s < 36, 13 >$  number is larger than 000-000001111111.111111111111B (127.9998779296875D) or smaller than 111-111110000000.000000000000B (-128D) the  $s < 10, 2 >$  number is saturated to 01111111.11B (127.75D) or 10000000.00B (-128D). Complex overflow counter 2 counts the number of saturations as if the signal amplitude was twice as big (when the  $s < 36, 13 >$  number is larger than 000-000011111111.111111111111B (255.9998779296875D) or smaller than 111-111100000000.000000000000B (-256D)).

The result of the counters is dumped into R2LC.SC1I, R2LC.SC1Q, R2LC.SC2I and R2LC.SC2Q after 4096 input samples. At the same time the saturation counters are restarted.

Input and outputs are at  $fr2ct$  rate.

No rounding is done here.

Parameters are TBD synchronous.

## 6.9 Noise - DC offset estimator

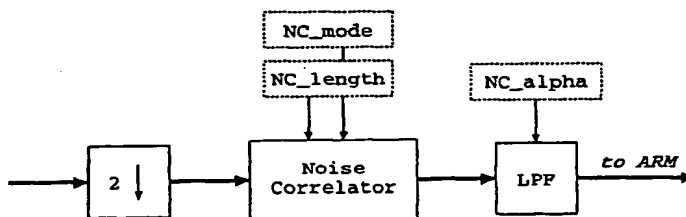


Figure 49: Noise - DC offset demodulator

This block has 2 functions : noise estimation and DC offset estimation. A filtered version of this value can be read by the ARM or other TBD hardware. This value could be used for setting thresholds in the acquisition hardware or DC removal filter. The function performed depends on the NC\_mode parameter.

NC_mode	function
0	noise estimation
1	DC offset estimation

The incoming samples at  $fr2ct$  rate are decimated with a factor 2. These samples are sent to the complex noise correlator. This block has 2 functions depending on the NC\_mode parameter :

In noise estimation mode the noise correlators calculate  $\sum_{i=0}^{NC\_length} |in|$ .

In DC offset estimation mode the noise correlators calculate  $\sum_{i=0}^{NC\_length} in$ .

$$0 \leq NC\_length \leq 32767$$

The noise correlators also decimate with a factor NC\_length+1.

TBD if we need a programmable down-shift after this correlator.

The values coming from the noise estimators are filtered by a simple low-pass filter :

$$filter_{out} = (1 - \alpha) * filter_{in} + \alpha * filter_{in} * z^{-1} \text{ TBC}$$

$$0 \leq \alpha < 1$$

By setting  $\alpha$  to 0, the filter is bypassed.

## 6.10 Acquisition hardware

This block will be discussed in detail in section 7.

## 6.11 Demodulators 0 and 1

This block will be discussed in detail in section ??.

60145130-022299

## 7 ACQUISITION HARDWARE

## 7 Acquisition hardware

This must be studied in more detail after final review of the UMTS acquisition needs. This block will contain at least : an NCO and downconverter for doing a serial search over different frequencies (TBC), a Chip Matched Filter with maximum codelength 256 and resolution 0.5 chip, a TBD FFT and UMTS specific hardware. Some outdated information can be found in document NL-UTRA-1.

< 'trick for FFT on modulated data. In stead of squaring (BPSK) or double squaring (QPSK) the data symbols to remove modulation, we could remove the data modulation by multiplying with  $e^{j*0}$  or  $e^{j*\pi}$  in case of BPSK or with  $e^{j*0}$  or  $e^{j*\pi/2}$  or  $e^{j*\pi}$  or  $e^{j*3\pi/2}$  for QPSK depending on the made data decisions. This is of course only possible if we have a data signal on which we can make data decisions. >

662220-924511059

## 8 DEMODULATORS 0 AND 1 STRUCTURE

## 8 Demodulators 0 and 1 structure

In most modes these 2 demodulators are used for handover between 2 cells, however they can also be used for other purposes. In the following paragraphs the demodulator structure will be explained in more detail. Figure 50 gives a general overview of one of the demodulators.

It consists of 3 tracking units (R, S and T) with their peripheral hardware like code generators and feedback signal generators like PED with PLL, TED with DLL, AED with AGC. This will be discussed in more detail later. There are 2 types of tracking units. Tracking unit R is of type 0, tracking units S and T are identical and of type 1.

Each demodulator also has a CCP block performing a combination of several incoming time-spaced chips. This block will also be discussed in more detail later.

Each of the 3 tracking units has the same input : the complex signal coming from the common level control. It is possible to track one signal source with one tracking unit. A signal source can be a physical transmitter or it can be a multipath component coming from one transmitter. So in one demodulator we can eg track 3 satellites or track 3 multipath components from a terrestrial base-station.

Roughly said there are 2 sort of configurations for the demodulator.

The first one is (like in UMTS) when the CCP is used. In that case mainly tracking unit R is used to receive upto 4 QPN channels. Tracking units S and T are of minor importance, they provide the possibility to receive an extra QPN channel each.

The second kind of applications does not use the CCP, tracking unit R will be used like tracking unit S and T. With this configuration it is eg possible to track independently 3 multipath components of one source and combine them in the symbol combiner.

Not all the hardware in fig 50 is used at the same time, this depends on the chip configuration. It must be possible to turn off some blocks to save power.

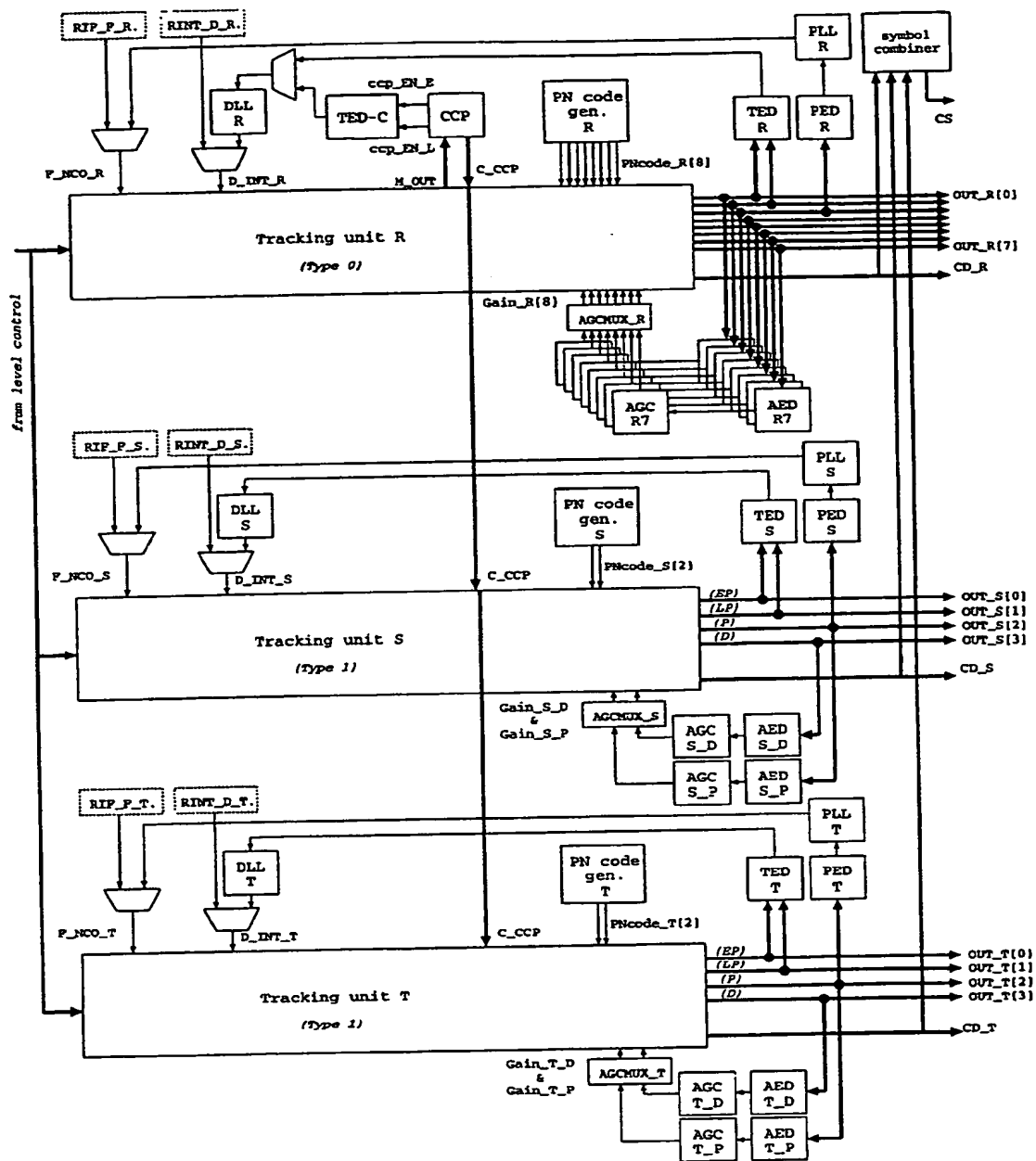


Figure 50: Demodulator overview

## 8.1 Tracking unit Type 0 structure

See fig 51.

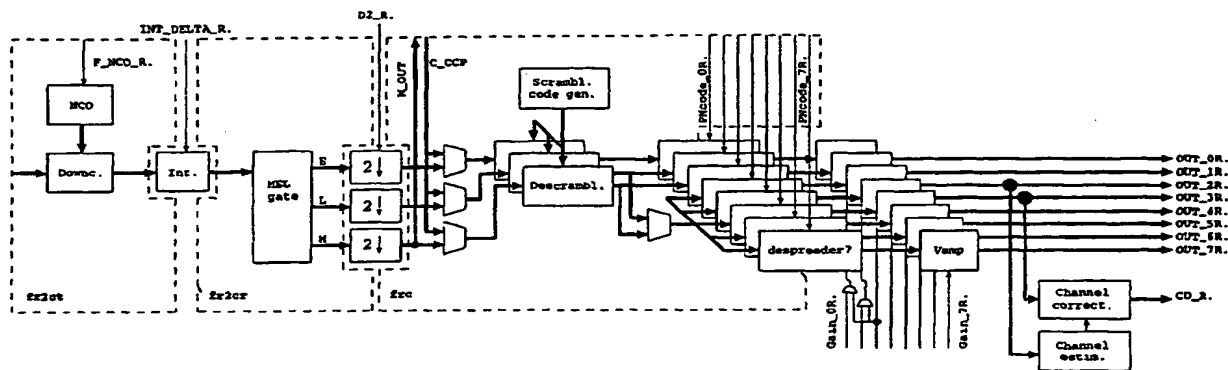


Figure 51: Tracking unit Type 0 structure

Tracking unit R is of type 0. This type of tracking unit can receive upto 4 chip synchronous QPN channels with different spreading factor  $s$  for each of the 8 codes. Due to the PNcode generation and common acquisition (TBC) there are some limitations : all symbols must have a common start moment and thus have a limited common multiple (TBC).

The unit can also be used to track a classic QPN signal with one pilot and (multiple) data. In this case OUT\_0 is the Early Pilot, OUT\_1 the Late Pilot, OUT\_2 the Middle Pilot. OUT\_3 upto OUT\_7 can be Middle Data symbols. In this mode the gain multiplexers should be set so that Vamps 0 and 1 are fed by Gain\_2 in stead of Gain\_0 and Gain\_1. All other codes have Vamps which are separately gain controlled.

The multiplexer before despreaders 3 to 7 is for OQPN purposes in which case the middle data does not come from the M output of the MEL gate but from the L output.



## 8.2 Tracking unit Type 1 structure

See fig 52

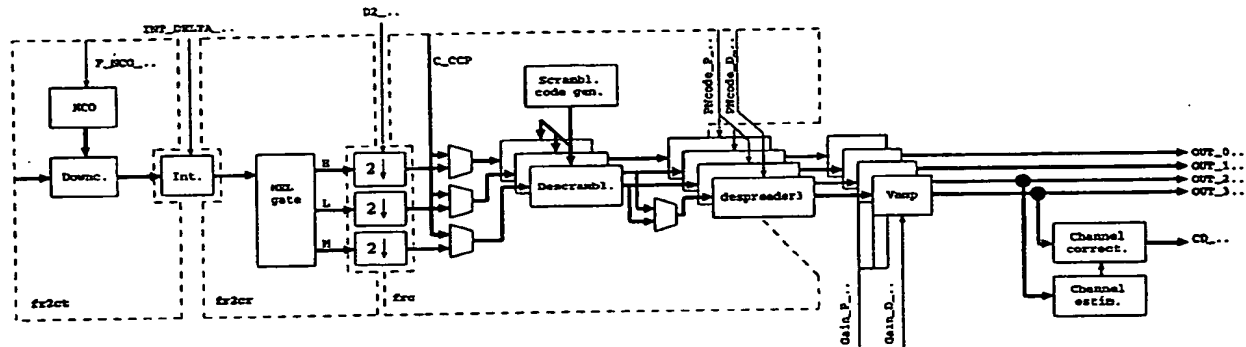


Figure 52: Tracking unit Type 1 structure

Tracking units S and R are of type 1. This type of unit is used to track a classic QPN signal with one pilot and one data stream. In this case OUT\_0 is the Early Pilot, OUT\_1 the Late Pilot, OUT\_2 the Middle Pilot and OUT\_3 the Middle Data symbols. There are 2 AGC loops, one for pilot, one for data.

Alternatively this unit can be used in conjunction with the CCP to receive one QPN channel.

The multiplexer before despreader 3 is for OQPN purposes in which case the middle data does not come from the M output of the MEL gate but from the L output.

## 9 Demodulator building blocks

This section describes the building blocks of the demodulator.

### 9.1 Downconverter and NCO

These blocks are used as actuator for the carrier phase/frequency tracking. A final downconversion is performed. See fig 53

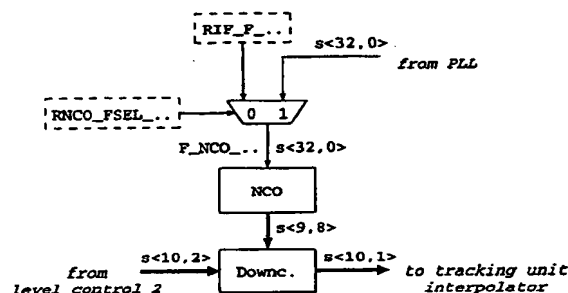


Figure 53: Tracking unit NCO and downconverter position

F\_NCO... sets the NCO frequency, this value can come from 2 sources selectable via the RNCO\_FSEL... parameter.

RNCO_FSEL...	NCO frequency controlled by
0	RIF_F... (ARM)
1	Hardware PLL loop

The NCO frequency must be ARM programmable when we use the CCP and thus no PLL. Another use is to close the carrier tracking loop in software.

This structure is present per demodulator and per tracking unit. Eg RIF\_F.S1, RNCO\_FSEL.S1 and F\_NCO.S1 are parameters and signals of tracking unit S of demodulator 1.

RNCO\_FSEL... and RIF\_F... are TBD synchronous.

#### 9.1.1 Tracking unit NCO

This block generates a cosine and sine value. The cos and sin are phase controllable by the ARM, phase step controllable by the ARM and frequency controllable by the ARM or PLL loop. See fig 54.

The sine and cosine values are generated with the 10 MSB of a  $s < 32, 0 >$  phase value (Eg RIF\_P.S1). The 8 LSB of this 10 bit number go to 2 lookup tables which contain the (256) values for sin and cos in  $[0, \pi/2[$  with a gain of 255/256. The lookup wordlength for sin and cos in quadrant 1 is  $u < 8, 8 >$ . The 2 MSB of the  $s < 32, 0 >$  bit phase register are used to recover the quadrant, sin and cos are  $s < 9, 8 >$  numbers. The output of the NCO is the complex signal  $(\cos + j.\sin)$ . cos and sin are signals in  $[-255/256:255/256]$ .

The phase of cos and sin can be directly controlled by the ARM by writing to the  $s < 32, 0 >$  bit phase register RIF\_P... and setting RIF\_F... to 0 and setting RIF\_S... to 0. For frequency control the F\_NCO... value is

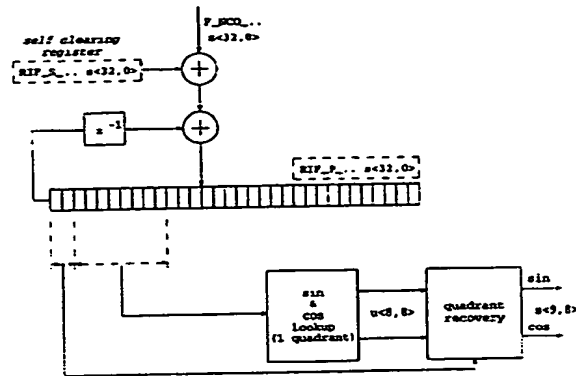


Figure 54: Tracking unit NCO

integrated with wrap around to get the RIF\_P... register. The F\_NCO... signal determines the frequency of the generated sine and cosine in the following way :

$$f_{sin} = f_{cos} = F\_NCO... / 2^{32} * fr2ct.$$

F\_NCO... can be ARM programmed directly by writing the value to RIF\_F... and setting RNCO\_FSEL... to 0. When RNCO\_FSEL... is set to 1, the F\_NCO... (and thus the sin and cos freq) comes from the hardware PLL of the tracking unit.

With RIF\_S... a one-time phase step can be given via the ARM. The register should be auto-cleared after applying it.

So the only hardware carrier tracking loop possible is a frequency controlled loop. A carrier phase change is done by temporarily changing the carrier frequency.

RIF\_P... and RIF\_S... are TBD-synchronous. RIF\_P... is only transferred when the ARM has written a new value to it, otherwise RIF\_P... is the wrap around integration of F\_NCO... (and RIF\_S...).

Input and output of this block is at fr2ct rate. This rate is the same in all tracking units.

< Acquisition block signals to NCO ??? (eg to set phase to 0 after acq) >

### 9.1.2 Tracking unit Downconvertor

If the input is  $X + jY$ , the output will be  $(X + jY) * (\cos - j\sin)$ . TBC that no other modes are needed. The computations are done full precision, the multiplications have 1 redundant bit as the most negative number will never be present in the sin or cos value. Thus the result of the multiplications are  $s < 18, 10 >$  numbers. This makes the full precision outputs  $s < 19, 10 >$  numbers. These full precision numbers are reduced to  $s < 10, 1 >$  numbers TBC, not as in common.

Input and output of this block is at fr2ct rate. This rate is the same in all tracking units.

## 9.2 Tracking unit Interpolator

This block is used as actuator for the chip phase/frequency tracking. Two modes are possible : in the first mode, tracking is done by adjusting the sample (chip) frequency every symbol. The second mode changes the sample (chip) phase every symbol. The feedback value can come either from the hardware DLL or from a ARM register.

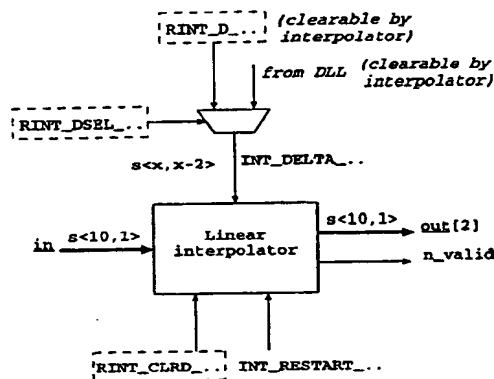


Figure 55: Tracking unit Interpolator

### 9.2.1 General interpolator principle

Linear interpolation between samples spaced approximately 0.5 chip is performed by :

$$\text{out}(k) = (1 - \text{INT\_MU}) * \text{in}(k-1) + \text{INT\_MU} * \text{in}(k)$$

where  $\text{in}(k-1)$  and  $\text{in}(k)$  are 2 consecutive equidistant samples at fr2ct rate. (Note that  $\text{in}(k)$  is not always the current input sample but is just the input sample coming after  $\text{in}(k-1)$ ). INT\_MU is a value in  $[0:1[$ .

The INT\_DELTA input is used to change INT\_MU by adding INT\_DELTA to the previous value of INT\_MU ( $\text{INT\_MU} = \text{INT\_MU} * z^{-1} + \text{INT\_DELTA}$ ). INT\_DELTA is limited to  $[-0.5:1]$ . When  $\text{INT\_MU} * z^{-1} + \text{INT\_DELTA}$  stays in  $[0:1[$ , for every input, one output is generated. However when  $\text{INT\_MU} * z^{-1} + \text{INT\_DELTA}$  would exceed the interval  $[0:1[$ , the value is wrapped back into  $[0:1[$  and 0 or 2 output samples are generated for the one input sample (plus some special action at the next sample). The n\_valid output of the interpolator block indicates the number of output samples generated (0, 1 or 2).

The following algorithm is used (C-program) (for a real input signal):

Note : INT\_MU is always a value in [0:1[ at the start of a new call of this function.

Note : in[2] is the current input sample, in[1] is the previous input sample, in[0] the input sample before in[1].

```
if(AddMuFlag==0)
{
    INT_MU = INT_MU + INT_DELTA;
}
AddMuFlag=0;
if((INT_MU>=0.0)&&(INT_MU<1.0))
{
    out[0]=in[1]*(1+INT_MU)+in[2]*INT_MU;
    n_valid=1;
}
else if(INT_MU<0.0)
{
    INT_MU=INT_MU+1.0;
    out[0]=in[0]*(1+INT_MU)+in[1]*INT_MU;
    INT_MU=INT_MU+INT_DELTA;
    out[1]=in[1]*(1+INT_MU)+in[2]*INT_MU;
    n_valid=2;
}
else
{
    INT_MU=INT_MU-1.0;
    AddMuFlag=1;
    n_valid=0;
}
```

out[0] must be processed before out[1] by the blocks following the interpolator.

The following table contains an example of outputs produced by the interpolator (real in,out) :

in	INT_DELTA	INT_MU used to calc out[0]	out[0]	INT_MU used to calc out[1]	out[1]	n_valid
1	0	0	0	—	—	1
2	0	0	1	—	—	1
3	0	0	2	—	—	1
4	0.2	0.2	3.2	—	—	1
5	-0.1	0.1	4.1	—	—	1
6	-0.25	0.85	4.85	0.6	5.6	2
7	0.1	0.7	6.7	—	—	1
8	0.4	—	—	—	—	0
9	0.25	0.1	8.1	—	—	1
10	0.4	0.5	9.5	—	—	1
11	0.5	—	—	—	—	0
12	0.3	0	11	—	—	1
13	0.5	0.5	12.5	—	—	1
14	0.5	—	—	—	—	0
15	0.5	0	14	—	—	1
16	0	0	15	—	—	1
17	1	—	—	—	—	0
18	0.1	0	17	—	—	1
19	1	—	—	—	—	0
20	1	0	19	—	—	1
21	-0.5	0.5	19.5	0	20	2
22	-0.25	0.75	20.75	0.5	21.5	2
23	-0.5	0	22	—	—	1
24	-0.5	0.5	22.5	0	23	2
25	-0.5	0.5	23.5	0	24	2
26	0.1	0.1	25.1	—	—	1
27	0	0.1	26.1	—	—	1
28	0	0.1	27.1	—	—	1
29	0	0.1	28.1	—	—	1
30	0	0.1	29.1	—	—	1

If INT\_DELTA is set to a fixed value the interpolator action results in a pseudo-sample rate change with a factor  $1/(1+INT\_DELTA)$ . The outcoming samples are not equidistant (when the incoming samples are). So the blocks after the interpolator must be designed so that they can follow this non-equidistant sample production. (Eg run a twice the nominal rate).

Note : instead of using interpolator hardware that returns 2 outputs at the same time and n\_valid with the values 0,1 or 2 and then serialising the outputs, it is maybe better to make an interpolator that runs at double speed with 1 output and a valid of 0 or 1.

## 9.2.2 Tracking unit interpolator operation

The interpolator complex data input is a  $s < 10, 1 >$  number. DELTA\_IN is a  $s < 10, 8 >$  number. The full range of this type is not used : DELTA\_IN is limited to  $[-0.5;1]$  1 included!.

If INT\_RESTART = 1, INT\_MU becomes 0 (and also the AddMuFlag in the algorithm). Otherwise INT\_MU is obtained by the algorithm explained before. So there is no direct INT\_MU control but can only be changed by INT\_DELTA (except by INT\_RESTART for setting INT\_MU to 0).  $INT\_MU * z^{-1} + DELTA\_IN$  is a  $s < 10, 8 >$  number. After wrapping back into  $[0;1]$ , the INT\_MU used for the multiplications is a  $u < 8, 8 >$  number.

The computations ( $out(k) = (1-INT\_MU)*in(k-1) + INT\_MU*in(k)$ ) are done full precision. (Note :  $(1-x)*a + x*b = a + x(b-a)$ , one multiplication less, but larger multiplication,  $b-a$  is  $s < 11, 1 >$ ). The full precision output is casted to a  $s < 10, 1 >$  (the same as the input wordlength TBD).

RINT\_D and the value coming from the DLL are produced at a rate smaller than the rate at which the interpolator runs. Eg the value from the DLL is produced at symbol rate. With RINT\_CLRD = 1, RINT\_D and the value coming from the DLL are cleared (to 0) after applying it.

RINT_CLRD	INT_DELTA clearing ?
0	INT_DELTA not cleared on read
1	INT_DELTA self clearing

So two chip phase/frequency tracking loops are possible : INT\_DELTA will get a new value (from the feedback loops) every symbol. In the first mode (RINT\_CLRD=0), we do a chip frequency tracking because INT\_DELTA has the same value during the complete symbol. In the second mode (RINT\_CLRD=1) we do a chip phase tracking. As INT\_DELTA is limited to  $[-0.5;1]$ , the chip frequency can be changed by a factor in  $[0.5;2]$  (RINT\_CLRD=0).

INT\_DELTA can come from 2 sources selectable via the RINT\_DSEL parameter : INT\_DELTA can be ARM programmed directly by writing the value to RINT\_D... and setting RINT\_DSEL... to 0. When RINT\_DSEL... is set to 1, the INT\_DELTA comes from the hardware DLL of the tracking unit.

RINT_DSEL...	Interpolator controlled by
0	RINT_D... (ARM)
1	Hardware DLL loop

The interpolator causes a delay of 1 sample (nominal). Eg when  $INT\_DELTA = cte = 0$ ,  $out = in * z^{-1}$  with a 0.0 added at the start.

The input samples are equidistant at fr2ct rate. The output samples of the interpolator are not equidistant at fr2cr rate. fr2cr is between  $fr2ct/2$  and  $2*fr2ct$ . So all the hardware after the interpolator must be designed to work at  $2*fr2ct$  although its nominal rate will be around fr2ct. fr2cr can be different in all tracking units at the same time.

### 9.3 MEL gate

The incoming stream at  $fr2cr$  is split in three streams at  $fr2cr$  rate.

$$E = in.z^{-2}$$

$$M = in.z^{-1}$$

$$L = in$$

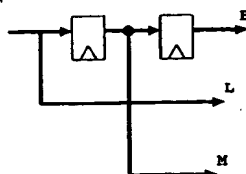


Figure 56: MEL gate

In this way each stream is spaced 0.5 chip.

The M signal of Tracking unit 0 is also used as input for the CCP block (see later).

All wordlengths are  $s < 10, 1 >$ .

### 9.4 Downsampling factor 2

A phase controllable downsampling with a factor 2 is performed here by skipping 1 incoming sample of 2 incoming samples. D2... (eg D2\_S1) defines which phase to skip.

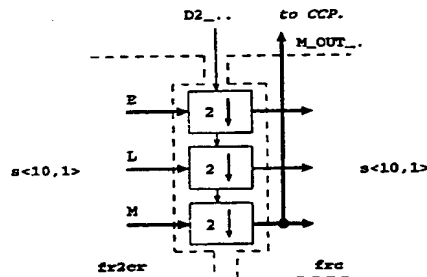


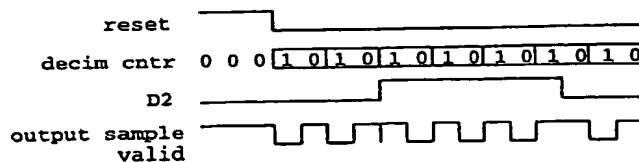
Figure 57: Downsampling factor 2

The output rate is  $fr = fr2cr/2$ . This clock is not equidistant and can be different in all the tracking units at the same time. The downsampled M outputs of tracking units R0 and R1 go to the CCP0 and CCP1 block (0 and 1 for the 2 demodulators).

*MAx-C-Mizer model :*

*the decimation phase is controllable as in fig 58.*



Figure 58: *MAx-C-Mizer* Downsampling factor 2 phase timing

During reset decimation counter is 0, also  $D2_{...}$  is 0. After reset the decimation phase counter restarts. A valid output sample is available when the counter is equal to  $D2_{...}$ .

### 9.5 chipstream selection

The 3 multiplexers allow to choose which signal goes to the final correlators. This can be the downsampled signal coming from the MEL gate or it can be the CCP output  $C\_CCP_{...}$ . This are all  $s < 10, 1 >$  signals. TBC-TBD.

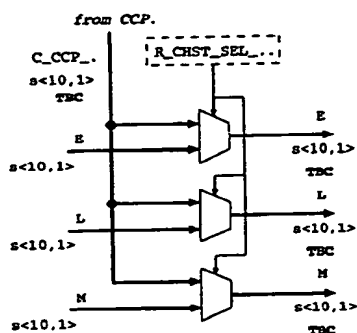


Figure 59: Tracking unit chipstream selection

When the  $C\_CCP$  is used as input the names  $E$ ,  $M$  and  $L$  are irrelevant.

There is a selection parameter per tracking unit :  $R\_CHST\_SEL_{...}$  (eg  $R\_CHST\_SEL_{S1}$ ).

$R\_CHST\_SEL_{...}$	Chipstream source
0	downsampled ELM gate
1	$C\_CCP_{...}$ from CCP.

### 9.6 Scrambling code generator and descramblers

Each tracking unit contains one scrambling code generator and 3 descramblers. The 3 descramblers are fed by the 3 streams coming from the chipstream selection block. All 3 descramblers in a tracking unit are fed by the same (de)scrambling code. See fig 60

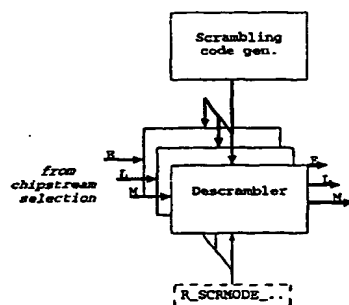


Figure 60: Tracking unit scrambling code generator and descramblers

#### 9.6.1 Tracking unit scrambling code generator

Functionally this is the same as the transmit scrambling code generator, but at frs rate and synchronisation might be different (TBD).

#### 9.6.2 Descrambler

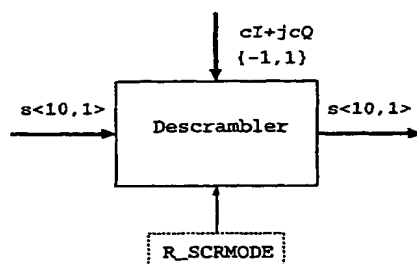


Figure 61: Tracking unit descrambler

Input data :  $dI+jdQ$  ( $s < 10, 1 >$ )

Input scrambling code :  $cI+jcQ$  ( $cI, cQ \in +1, -1$ )

This block should have 3 modes selectable by  $R\_SCRMODE...$  (eg  $R\_SCRMODE\_S1$ ) :

R_SCRMODE...	mode
00	Off
01	Complex Descrambling
10	Dual Real Descrambling

- Off : output = input , both  $s < 10, 1 >$ .
- Complex descrambling : output =  $(dI + jdQ)/(cI + jcQ) = (dI.cI + dQ.cQ + j(-dI.cQ + dQ.cI))/2.0$   
All calculations are done full precision (giving a  $s < 12, 2 >$  number) and then casted to a  $s < 10, 1 >$  number. There are some cases in which we must saturate instead of a simple casting. Eg when dI and dQ are the most negative number in a  $s < 10, 1 >$  and cI and cQ are -1, or when dI and dQ are the most negative number in a  $s < 10, 1 >$  and cI = -1 and cQ = 1, we saturate to the most positive value in the  $s < 10, 1 >$  output.
- Dual real descrambling : output =  $dI/cI + j dQ/cQ$   
Calculations are done full precision (giving a  $s < 11, 1 >$  number) and then casted to a  $s < 10, 1 >$  number. There are some cases in which we must saturate instead of a simple casting. Eg when dI is the most negative number in a  $s < 10, 1 >$  and cI is -1, or when dQ is the most negative number in a  $s < 10, 1 >$  and cQ = -1, we saturate to the most positive value in the  $s < 10, 1 >$  output.

< Check if all applications are possible (UMTS/IS-95,...). Is there a use for the dual real scrambling ??? Needs a lot of despreaders TBI >

In the 3 modes the delay between in and output should be the same. Input and output is at frc rate.

662220-9244103

## 9.7 Tracking units Despreaders

### 9.7.1 Tracking units Despreaders overview

The number of despreaders in a tracking unit depends on the tracking unit's type. See fig 62 and fig 63.

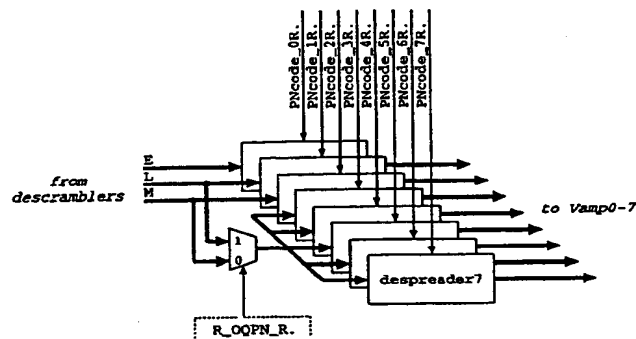


Figure 62: Tracking unit type 0 despreaders overview

Tracking units of type 0 (R) contain 8 despreaders that despread a complex signal with a real code. Each desreader is fed by a different spreading code. In this way we can despread 4 QPN stream in one tracking unit of type 0. Desreader 0 despreads the complex E signal coming from the E descrambler. Desreader 1 despreads the complex L signal coming from the L descrambler. Desreader 2 despreads the complex M signal coming from the M descrambler. Despreaders 3 to 7 all despread the same signal : this can either be the M or the L signal. The selection is done with the R\_OQPN\_R. parameter.

R_OQPN_R	Type 0 despreaders 3-7 input
0	descrambler M output
1	descrambler L output

Tracking units of type 1 (S-T) contain 4 despreaders that despread a complex signal with a real code. Despreaders

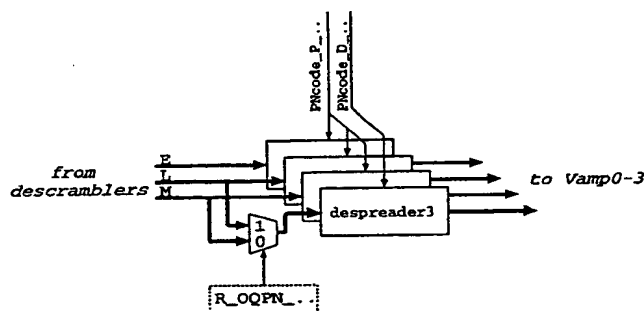


Figure 63: Tracking unit type 1 despreaders overview

0 to 2 all use the same spreading code (PNcode\_P...), desreader 3 uses a different spreading code (PNcode\_D...). In this way we can track 1 QPN stream by desreading an Early Pilot, Middle Pilot, Late Pilot and Middle

Data in one tracking unit of type 0. Despreader 0 despreads the complex E signal coming from the E descrambler. Despreader 1 despreads the complex L signal coming from the L descrambler. Despreader 2 despreads the complex M signal coming from the M descrambler. Despreader 3 can either despread the M or the L signal. The selection is done with the R.OQPN... (eg R.OQPN.S1) parameter.

R.OQPN...	Type 1 despreaders 3 input
0	descrambler M output
1	descrambler L output

Despreading the L signal in stead of the M signal in depreaders 3-7 or 3 is done when the transmit stream is an OQPN stream where the Data is delayed 0.5 chip with respect to the Pilot (not the opposite!).

### 9.7.2 Despreaders detailed description

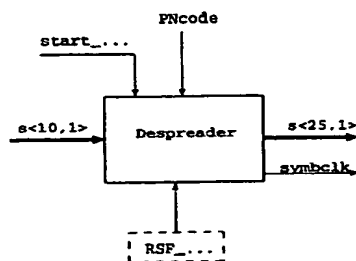


Figure 64: Tracking unit Depreaders

The incoming samples are multiplied with the PNcode, where 0B is +1D and 1B is -1D. These numbers are integrated over the spreading length with a certain phase and then an output symbol is generated.

In tracking units of type 0, each of the 8 despreaders can have a different spreading factor. In tracking units of type 1, despreaders 0-2 have the same (pilot) spreading factor, the (data) spreading factor of despreaders 3-7 can be different.

The spreading factor is programmed with the RSF.... parameter. (In type 0 tracking units, the ... stand for despreaders number, tracking unit, demodulator. Eg RSF\_2R1 sets the spreading factor of despreaders 2 in tracking unit R of demodulator 1. In type 1 tracking units, the ... stand for Data/Pilot, tracking unit, demodulator. Eg RSF\_PS0 sets the spreading factor of the Pilot despreaders (3) in tracking unit S of demodulator 0).

The spreading factor is  $RSF.... + 1$ .

RSF.... is a  $u < 15, 0 >$  signal. So the maximal spreading factor is 32768, non-CDMA mode is done by setting RSF.... to 0.

The accumulator is a  $s < 25, 1 >$  number. Overflow is only possible when the spreading factor is 32768, all PNbits are -1D and all incoming samples are -256D. We assume that this will never occur and thus the accumulator is full precision and overflow free with this assumption.

With the start.... signal the despreaders can be restarted in the middle of a symbol. TBD this signal.

A symbolclk signal is generated every symbol to indicate the end of the integration. TBD if phasesteps are needed.

## 9 DEMODULATOR BUILDING BLOCKS

The incoming chips are at frc rate, which is not equidistant, so non-equidistant symbols can be produced. We can have up to  $2*(8+2+2) = 24$  different symbol rates at the same time in the ASIC.

*MAx-C-Mizer model (without phasesteps) :*

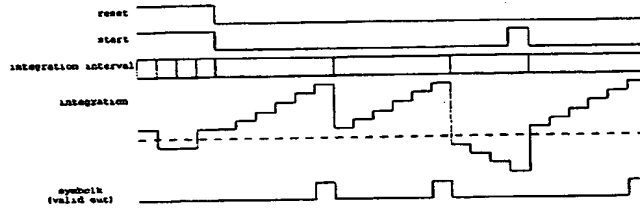


Figure 65: MAx-C-Mizer despreaders phase timing

After  $start==1$  (or reset) a new integration starts. A symbolclk is generated together with the last integrated chip. At that time a new symbol output is present. So the symbolclk serves as 'valid bit' for the blocks running at symbolrate following the despreaders. During reset symbolclks are suppressed to 0.

### 9.8 Variable amplifiers

This block is used as actuator for the signal amplitude tracking. Structure depends on wordlength coming from despreaders. Each Vamp can have a different gain.

The output of the Vamps are the complex soft symbols OUT[0] to OUT[3].

Note : It might be interesting to incorporate a module that returns hard bits based upon the soft symbols. (TBD)

All Vamps can run at a different rate.

All wordlengths are TBD.

### 9.9 Channel estimation

This block can be used to make a channel estimation (TBD algorithm) based upon the soft OUT[2] output. The output of this block is at OUT[2] symbolrate.

Note : NOT used for UMTS.

### 9.10 Channel correction

This block performs a TBD channel correction on the OUT[3] data with the aid of the channel estimation for that tracking unit. The output of this block is CD at the same symbolrate as OUT[3].

### 9.11 PNcode generators

These blocks generate the complex PNcodes for the despreaders. Structure is TBD.

Tracking unit R is equipped with a PN code generator which generates 4 complex codes, units S and T have a generator which generates 1 complex code.

### 9.12 AED and AGC

The AED is the error detector for the signal amplitude tracking. The AGC does a filtering on this signal and outputs the signal going to the Variable amplifiers.

Tracking unit R has 8 separate AED and AGC for each despreader in the tracking unit, while tracking unit S and T only have 2 AED and AGC.

AED and AGC are TBD.

Note that not all AED and AGC HAVE TO BE the same, eg AGCR0, AGCR1 and AGCR4 to AGCR7 can be simpler AGCs as they must not normalise the signal for a DLL, but just to maximase the number of bits at the output. (TBC).

### 9.13 PED and PLL

The NCO of each tracking unit can be set by an external block like ARM software or can be controlled by the PLL. The PED works on the OUT[2] signal.

When the CCP is used, the PLL and PED can be turned off.

PED and PLL are TBD.

### 9.14 TED and DLL

The TEDs are used as error detectors for the chip timing tracking. TED-C is used when the CCP is used as a signal source for the despreaders of the unit, while TEDR, TEDS and TEDT are used when classic Early-Late correlator tracking is done. The output of the TED go to the DLLs, chip frequency controlling the interpolator.

TED and DLL are TBD.

### 9.15 symbol combiner

When the 3 tracking units are used for tracking different multipaths of the same signal, a hardware combination of the 3 CD ouputs can be performed. Functionaly this is only an addition of the complex CD numbers. However the symbol timing of CD\_R, CD\_S and CD\_T will be different which will complicate the coherent symbol combining.

### 9.16 CCP - Chip Combining Part

This block performs a weighted coherent combination of 8 chipstreams, spaced 1 chip, into one new chipstream. To combine them weighted coherently a channel estimation (ampitude, phase) of each of the 8 chipstreams is made.

This block is discussed in detail in section 14.

## 10 DEMODULATOR LOOPS

### 10 Demodulator loops

The demodulator loops will be discussed in more detail here.

#### 10.1 Carrier phase/frequency tracking

TODO

*only frequency control*

#### 10.2 Chip phase/frequency tracking

TODO

#### 10.3 Symbol amplitude tracking

TODO

66220-52454103



# 11 Demodulator configured to track 3 QPN sources

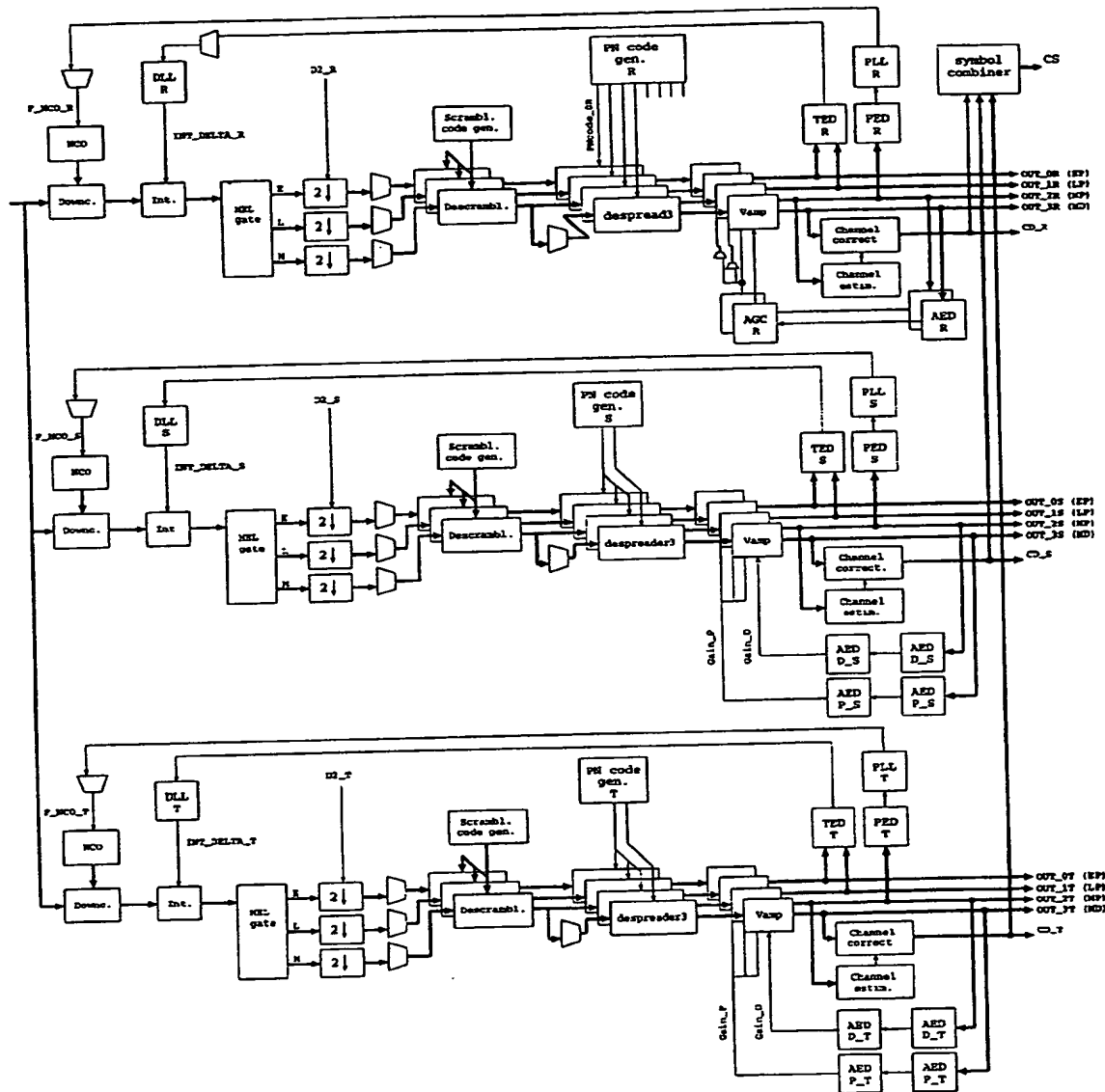


Figure 66: Demodulator used to track 3 QPN sources

The QPN sources can be 3 different transmitters or 3 multipaths from one transmitter. When the 3 tracking units each track one multipath component of the same transmitter, the CD\_R, CD\_S and CD\_T can be combined into CS.

662220-92454709

## 12 Demodulator configured to track 3 OQPN sources

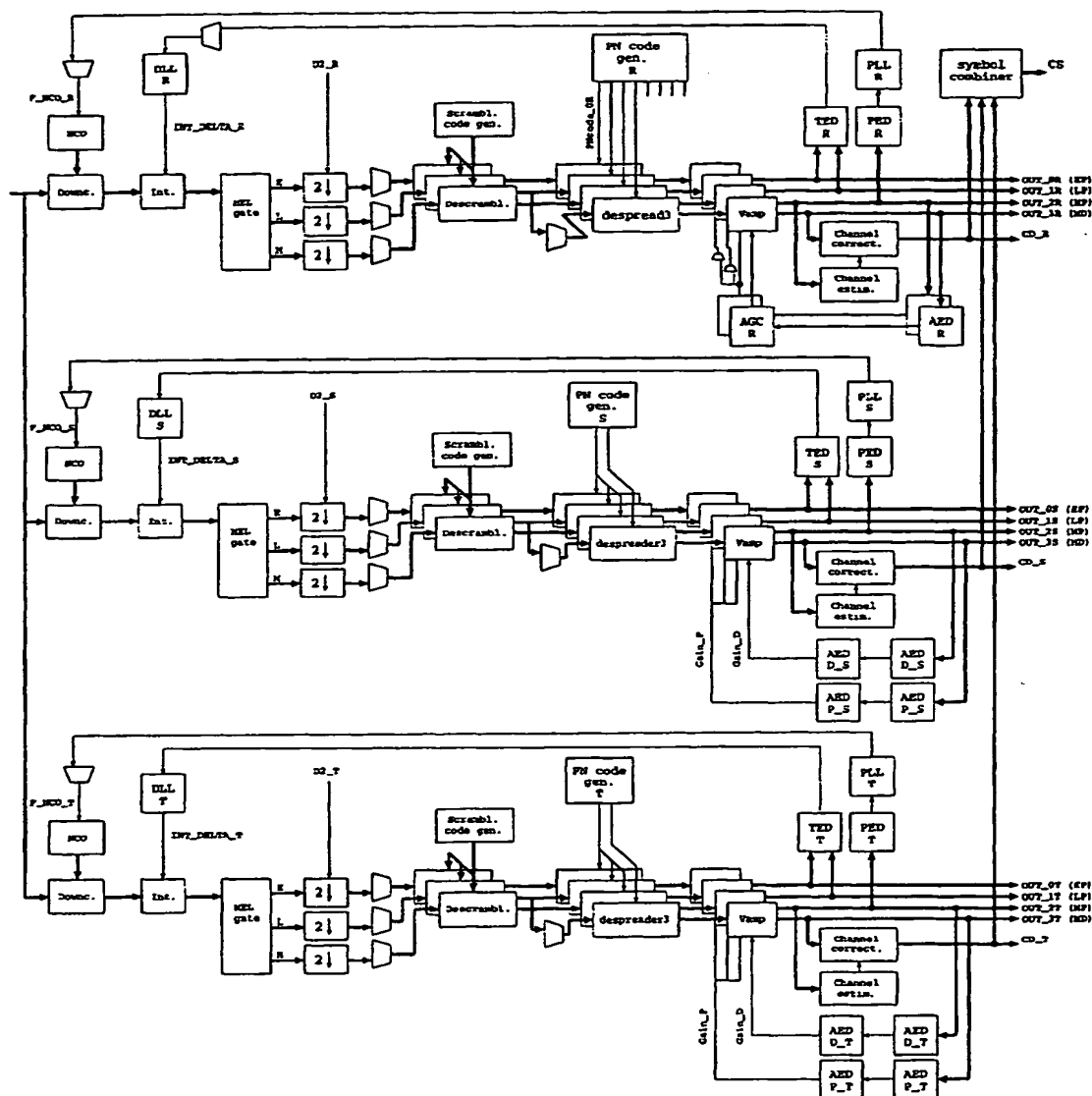


Figure 67: Demodulator used to track 3 OQPN sources

July 12, 1999

**12 DEMODULATOR CONFIGURED TO TRACK 3 OQPN SOURCES**

**Revision 9-n Preliminary**

The OQPN sources can be 3 different transmitters or 3 multipaths from one transmitter. When the 3 tracking units each track one multipath component of the same transmitter, the CD.R, CD.S and CD.T can be combined into CS.

60445426-072399

This section gives a detailed explanation on the use of the demodulator as a receiver where multipath components are coherently combined at chiprate. The CCP block of the demodulator is only used in this mode and is also discussed in detail in this section. Fig 68 gives an overview of the demodulator configuration and used blocks in CCMR mode.

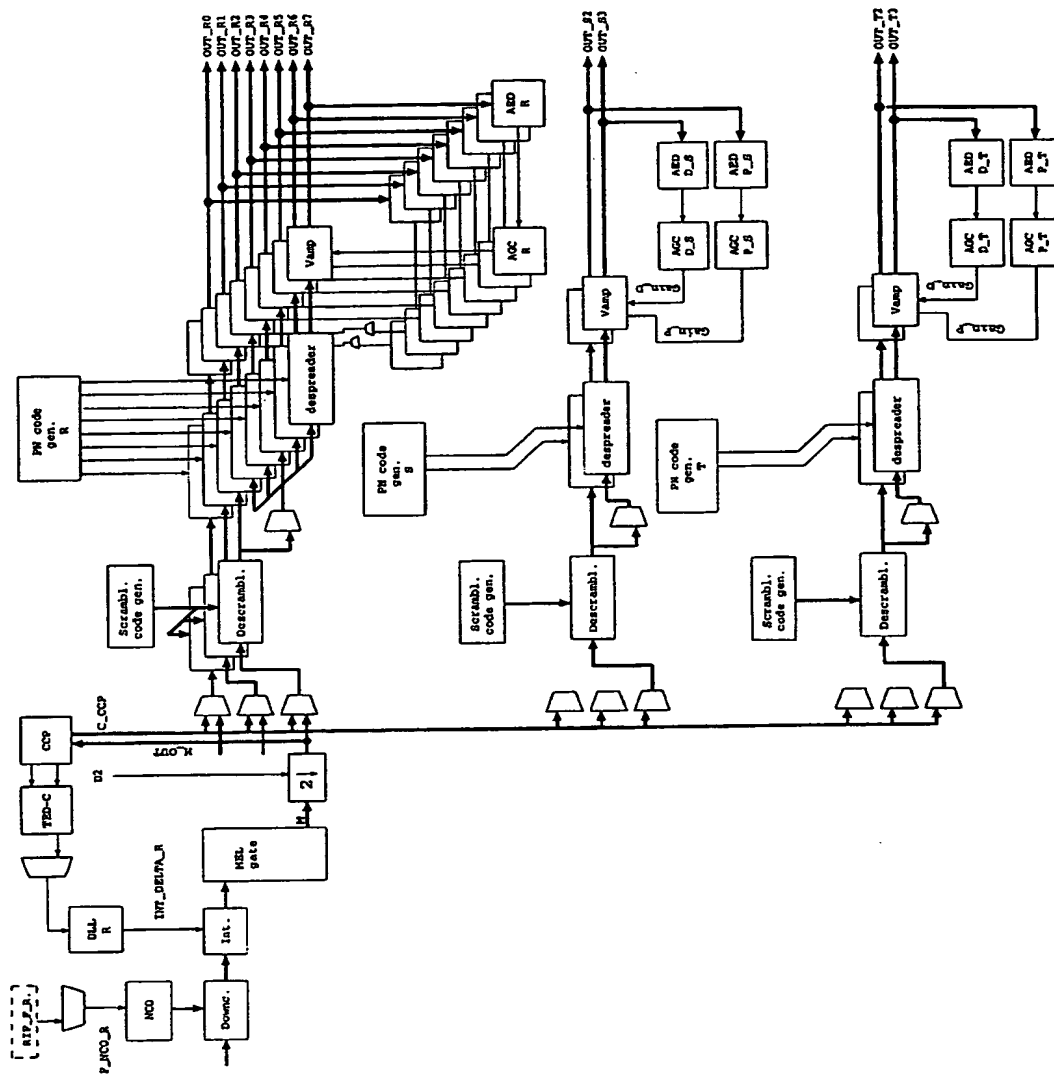


Figure 68: *Demodulator used as CCMR*

This specification is written to receive a UMTS waveform, so not general. Maybe some things must be more flexible (TBD). The most 'fixed' UMTS parameter in the CCP/CCMR is the slotlength of 2560 chips and the SF going from 4 to 512 with 2560/SF integer. So 5 to 64 symbols in 1 slot.

The CCMR configuration reuses almost everything from the tracking unit except for the PLL and PED and the tracking units specific TEDs. A large extra block that is not used when using Early-Late correlator tracking is the CCP. So roughly said the CCMR exists out the CCP that generates a new chipstream from the incoming chipstream and the classic descrambler, despreaders, ... hardware. The NCOs in the tracking units are set to a 'fixed' value by programming RIFF\_R0, RIFF\_R1.

With the configuration of fig 68 it is possible to receive 6 QPN channels. These 6 channels must be synchronous as they use the same CCP. From these 6, 4 QPN channels must also have the same scrambling code. With tracking unit S and T with CCP as input we can receive 2 QPN channels with a different scrambling code (if necessary). To receive two asynchronous transmitters we must use the 2 demodulators.

If only 4 QPN channels must be received, tracking units S and T can be turned off.

The only despreading in the CCP is the pilot symbol despreading used to make the channel estimations. Chip phase tracking is done by timing error detector TED-C and DLL\_R working at slotrate, so not at symbolrate (TBC). Closing at symbolrate is not possible as we have no early and late energies at symbolrate anywhere in the CCMR. (Pilot symbols are not continue and for the data despreaders we have only one chipstream)

More details can be found in section 14.

## 14 CCP overview

This part performs the coherent combination of 8 chipstreams, each spaced 1 chip, into one new chipstream. We have 8 fingers fixed spaced 1 chip, where a channel estimation is done for that chip phase with the aid of pilot symbols. This channel estimation is used to 'correct' the chipstream of the respective finger, after which all fingers can be combined. MRC with optional zero forcing is used to combine the different chip phases. Decision aided CCMR only, the pilot symbols can have a SF from 4 to 512 and may be arbitrarily distributed over the slot. See fig 69.

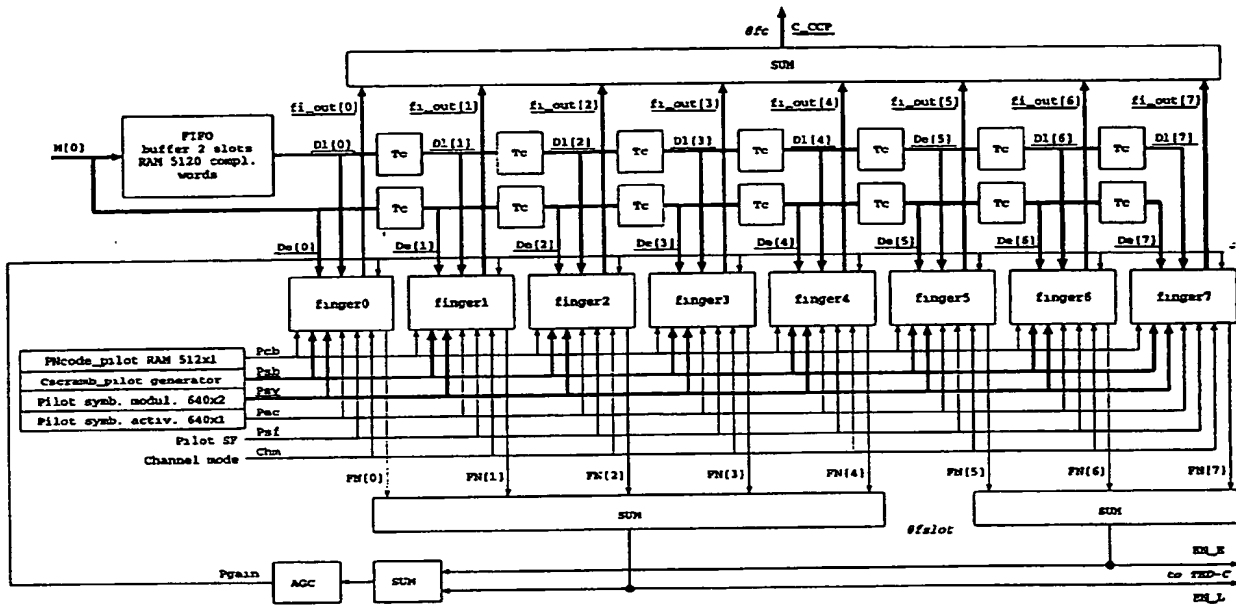


Figure 69: CCP overview

Fingers 0 to 4 contribute to the Late multipaths, fingers 5 to 7 to the Early multipaths (TBC). Note that there is no real 'Middle' finger, this means that in the case of a single path, the correlation energy will be split over finger 4 and 5 and we will never correlate at the 'top' of the correlation shape.

The MCR will be initialised so that the strongest peak will be between finger 4 and 5. With the phase controllable decimation (D2) the chipphase can be set with a resolution of 1/2 chip.

Very important : after acquisition, the chipphase of the strongest peak will be known with a resolution of 0.5 chip. When going to tracking mode, the D2 parameter and the correlators symbol edges (start of integration) must be set correctly. All pilot despanders in the CCP are symbol synchronous. All despanders after the CCP are also synchronous with the pilot despanders at slotrate and at symbolrate when they have the same SF as the pilot despanders. For setting the D2 and start of integration of the despanders after acquisition see fig 70

Suppose that from the acquisition process it is known that the M sample leaving the interpolator is the first sample that should be integrated by a simple correlator to get the optimum timing from 1 despander. For synchronising the CCMR we will decimate so that NOT the M sample but the E and L samples are outputs of the decimator

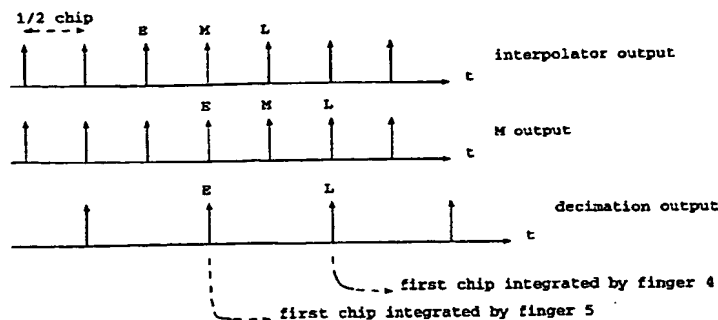


Figure 70: Correlators synchronisation with the incoming chipstream

and we start the CCMR integrators so that in finger 4 the first sample integrated is sample L and in finger 5 the first sample integrated is sample E.

Be very careful when going from acquisition to tracking (eg interpolator has 1 sample delay TBC).

The chips coming from decimator are sent to a FIFO with length of 2 slots. The function is to delay the incoming chips with 2 slots because the channel estimation process takes 2 slots time. (There is only a channel estimation for the incoming chip after a time of 2 slots).

This delay could be done by having a complex RAM with 5120 complex words (1 address of the RAM with containing 1 complex value) and a simple cyclic address counter going from 5119 to 0. The value at the address is FIRST read, afterwards at the same address the new value is written. (Any other hardware could be used with the same functionality).

Approximatively  $5120 * 2 * 10 = 102400$  bit RAM is needed.

Each finger has as inputs :

- Pcb : the codebit for despreading the pilot chipstream. The spreading code is stored in a RAM of 512 bits. This is a real signal, no QPN pilot is possible (TBC).
- Psb : the complex descrambling bits coming from the descrambling code generator.
- Psy : the data modulation on the pilot symbols. How this is stored is TBD. We could use a RAM to store the modulation of a complete slot, so we need a RAM of 640x2 bits. When a higher SF is used not all 640 locations will be used. Eg with SF 256 only the first 10 locations of the RAM will be used. Pilot modulation can change on a slot to slot basis (TBC-PR) so take provisions to make this possible (double buffering,...).
- Pac : activity bit for pilot symbols. This eliminates the need for having the pilot portion as a continuous portion at the beginning of the slot. (Remark Sanchez : ARIB probably pilots at end of slot). Again a RAM of 640x1 could be used.
- Psf : The pilot SF.
- Chm : channel mode parameter, selects the algorithm to use to make the channel estimations. (slow fading : 0, fast fading 1).
- Other configuration inputs like : threshold de decide on which finger there is a signal, filter coefficients for channel estimation filtering,  $1/(\text{number of pilots in a slot})$  etc. They are not on fig 69 as they are too detailed for this drawing.



## 14 CCP OVERVIEW

NOTE : the RAMs of 640 bits could be smaller if it is not required that we could have a burst of pilot chips equal to 4 chips anywhere in the slot. It is most likely that this is not required. Eg 8 consecutive pilot symbols SF can be replaced by 1 pilot symbol with SF 32. TBD what we do !!! For the moment I use the 640 bits in the simulation model.

Each finger has a complex  $C\_CCP[x]$  output at chiprate. This is the delayed chips multiplied with the complex conjugate of the channel estimation of finger x. Each finger also has a  $FNx$  output at slotrate which is the energy of the coherent accumulation of all pilot chips/symbols in a slot of finger x. The sum of all  $FNx$  is calculated and goes to the pilot AGC. In this way  $C\_CCP$  will not be dependent on the pilot energy.

As we have fixed finger spacing we only need a global DLL. The DLL will work on slotrate, the Late and Early energys are calculated as :  $EN\_L = FN0+FN1+FN2+FN3+FN4$ ,  $EN\_E = FN5+FN6+FN7$ .  $EN\_L$  and  $EN\_E$  go to the DLL which feedbacks to the interpolator at the input of the CCMR.

An AFC or PLL is not included.

The CCMR will have a algorithmical latency of 2 slots.

## 14.1 CCP finger

This section describes one of the 8 fingers. See fig 71.

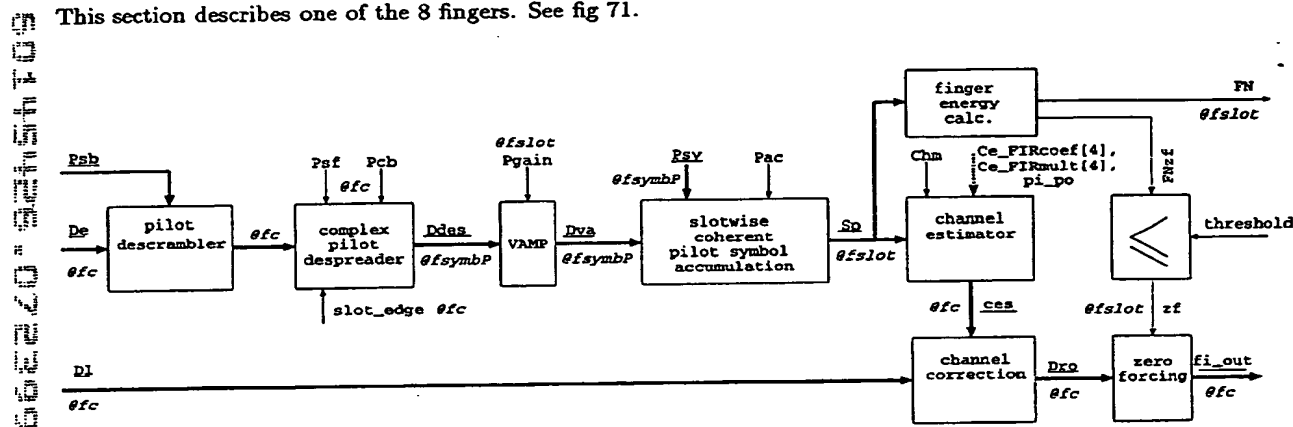


Figure 71: one CCP finger

## 14.1.1 Descrambler

The incoming chips are descrambled with  $Psb$ . This code and its phase is common for all fingers. The phase has to be set during an acquisition process initialising the CCMR. Has the same functionality as the other descramblers.

## 14.1.2 Complex pilot despreader

The complex signal coming from the descrambler at chiprate is despread with the pilot PNcode ( $Pcb$ ), only 1 despreader, so the pilot must be a QPSK or BPSK signal. The pilot PNcode has a PNlength of  $Psf$ .  $4 \leq Psf \leq 256$ , and  $k * Psf = 2560(TBC)$  with  $k$  a positive integer.

The despreader works continuously and is synchronised to the slot edge at chiprate. This means that a new symbol starts at the start of the slot ( $slot\_edge=1$ ).

### 14.1.3 Variable Amplifier

The complex symbol coming from the despreader is sent through the Variable Amplifier (VAMP). The complete CCMR has one global AGC which sets the Pgain at slotrate.

For different spreading factors, the initial gain must be set to a different value, eg to 1.0 for SF 512, to 128.0 for SF 4. (TBC)

### 14.1.4 Slotwise coherent pilot symbol accumulation

In this block a coherent pilot symbol accumulation is done on a slot by slot basis. The Pac input defines if the symbol coming from the VAMP is a pilot symbol. See fig 72.

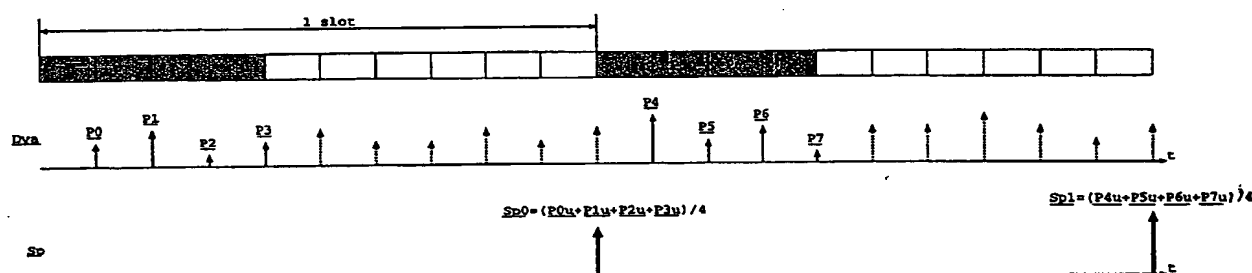


Figure 72: Slotwise coherent pilot symbol accumulation

In this example the Psf is 256, Pac would be 111100000....0000.

$P_i$  are the complex despread pilot symbols  $D_{va}$  (@fsymbB). In order to accumulate them coherently, the pilot modulation must be removed first. This modulation is known a priori and must be present at the  $Psy$  input. For QPSK  $Psy$  can take 4 values: +1, -j, +j, -1. For QPN  $Psy$  can take 2 values: +1 and -1. So  $Psy$  is represented by a 2 bit value ( $Psy[0]$  and  $Psy[1]$ ). The values  $P_i$  are then demodulated in the following way ( $P_{iu}$  are the demodulated values of  $P_i$ ) (TBC):

$Psy[0..1]$	$P_i$	$P_{iu}$
00	$P_{i,i} + j * P_{i,q}$	$P_{i,i} + j * P_{i,q}$
01	$P_{i,i} + j * P_{i,q}$	$-P_{i,q} + j * P_{i,i}$
10	$P_{i,i} + j * P_{i,q}$	$P_{i,q} - j * P_{i,i}$
11	$P_{i,i} + j * P_{i,q}$	$-P_{i,i} - j * P_{i,q}$

For QPN  $Psy$  must only take the values 00 or 11.

$Sp_j$  are the complex accumulation of these demodulated pilot symbols from the current slot, divided by the number of pilots (or multiplied by 1/number of pilot symbols):

$$Sp = \frac{\sum_{i=0}^{n-Psf} P_{iu}}{\#pilots}$$

This is equivalent with despreading over all the pilot chips in the slot in the case of unmodulated pilot symbols.

$Sp$  values are generated at slotrate fslot. The value is available at the end of the slot.

This module must be slot-synchronous.

## 14 CCP OVERVIEW

## 14.1.5 Finger energy calculation

## TBD-TBC

Here a measure for the finger energy is calculated slot by slot. See fig 73

The energy is calculated as follows :  $Sp_i^2 + Sp_q^2$ .

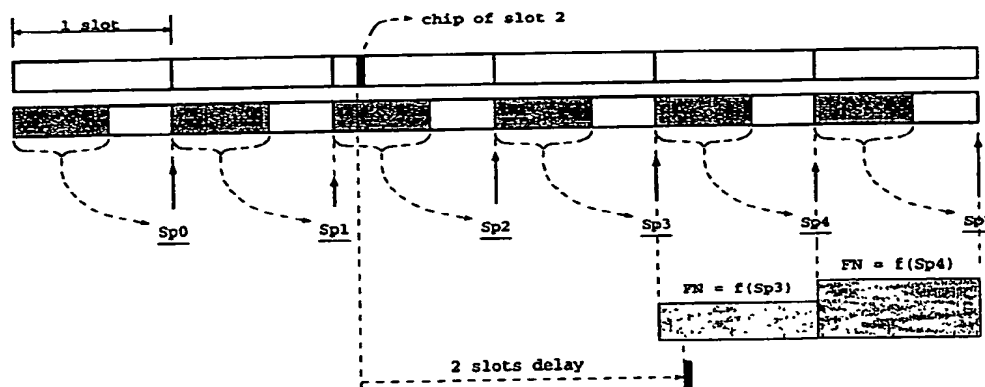


Figure 73: Finger energy calculation

This energy will be used for the DLL. For the zero forcing we also need the finger energy but with an extra delay :  $FNzf = FN \cdot z^{-1}$ .

In this way there is a minimal delay in the DLL loop and the zero forcing is coherent with the slots.

< We only need a slow DLL as with a maximum doppler frequency of 1kHz on 2GHz carrier and a 3ppm MS Xtal, with 4.096 Mcips/sec the maximum shift in chipphase is :  $(1\text{kHz} \cdot 4.096\text{MHz}) / 2\text{GHz} + 3\text{ppm} \cdot 4.096\text{MHz} = (2.048 + 12.288) \text{ chips/sec} = 14.336 \text{ chips/sec}$ . This is less than 0.009 chips/slot. >

## 14.1.6 Channel estimator

This block performs a filtering or interpolation on the  $Sp$  values. The exact function to perform depends on the Chm (channel mode) input (fast or slow fading channels). The output of this block is the channel estimation ces at chiprate. When Chm = 0, the Ce\_FIRcoef[4] and Ce\_FIRmult[4] inputs are needed, when Chm = 1 the pi\_po input is needed.

## Channel mode 0: Slow fading

In this mode ces is constant over a complete slot. ces is a filtered versions of the incoming  $Sp$  values. See fig 74.

The multiplication after the filter is to have a FIR filter with unity gain. To avoid a transient in the amplitude on the signal coming from the filter, 4 different values are stored for this gain. The first output of the filter gets gain Ce\_FIRmult[0], the second output Ce\_FIRmult[1], the third Ce\_FIRmult[2] and Ce\_FIRmult[3] is used on sample number 4 leaving the filter and in steady state mode. (TBC if needed).

All filter taps should be initialised to 0 at the start of the process.

The filter and multiplier work at slotrate fslot, ces are samples at chiprate. (oversampling of filter output)

See fig 75 for an overview of the CCP finger process in the case of channel mode 0.

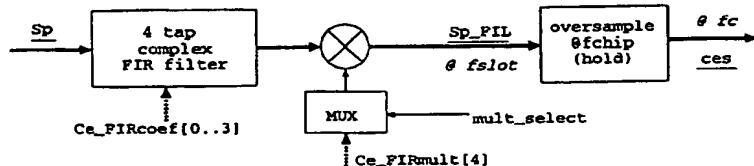
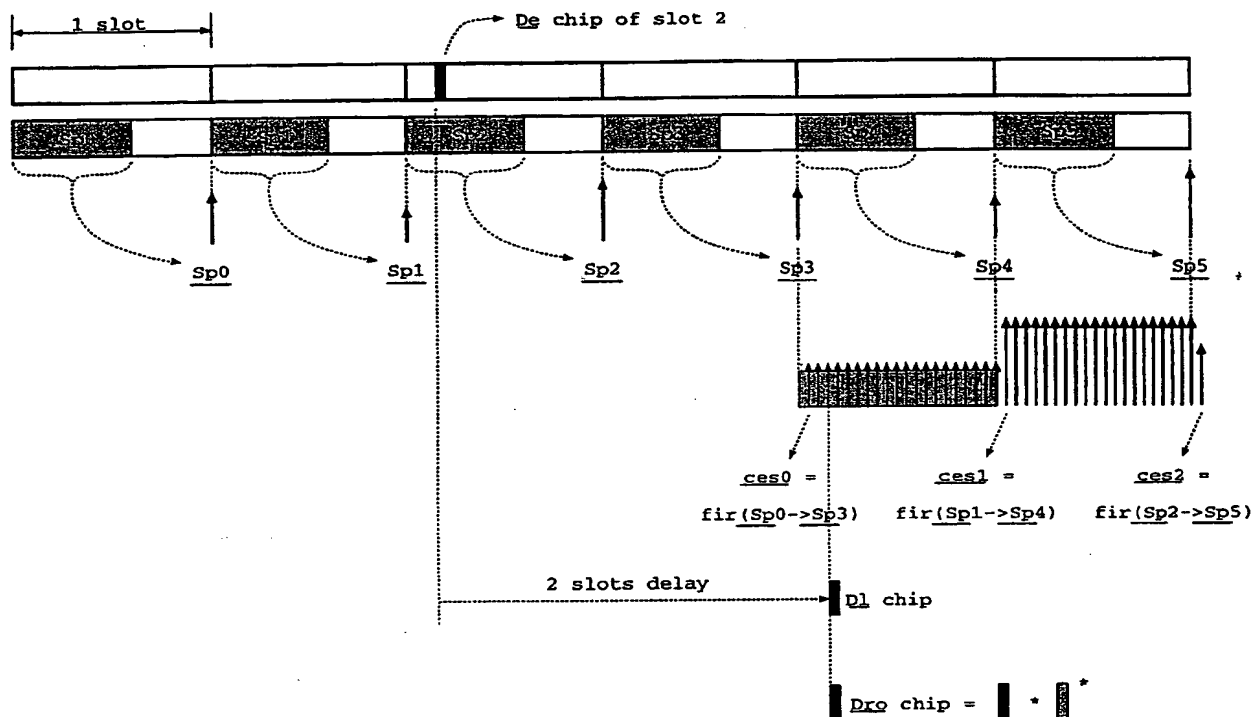
Figure 74: *Slow channel estimation filter*

Figure 75: CCP finger process in the case of Slow fading mode

The different pilot symbols are demodulated and coherently accumulated giving the values  $\underline{\text{Sp0}}$  to  $\underline{\text{Sp5}}$ . The channel estimations  $\underline{\text{ces}}$  are the output of the 4 taps FIR filter,  $\underline{\text{ces0}}$  is a function of  $\underline{\text{Sp0}}$  to  $\underline{\text{Sp3}}$ .  $\underline{\text{ces0}}$  is constant over slot number 4. The  $\underline{\text{De}}$  chip from slot 2 is delayed 2 slots so that it is available with slot 4 as  $\underline{\text{Dl}}$  chip. This chip is multiplied with the complex conjugate of  $\underline{\text{ces0}}$  to give the  $\underline{\text{Dro}}$  chip of this finger.

It is clear that the chip arriving in slot 2 is 'corrected' with the info from pilot symbols of slot 0,1,2 and 3. Every chip is always corrected with the aid of the Before Before, Before, Present and After slot. (unless some filter taps are set to 0). Channel estimations change only at slotrate. Note that Sp3 is generated together with the last chip of slot 3 while ces0 which is a function of Sp3 is used for all chips of slot 4.

**Channel mode 1: Fast fading**

In this mode ces are interpolated values between the current and the previous Sp values entering the channel estimator. So ces changes at chiprate. See fig 76

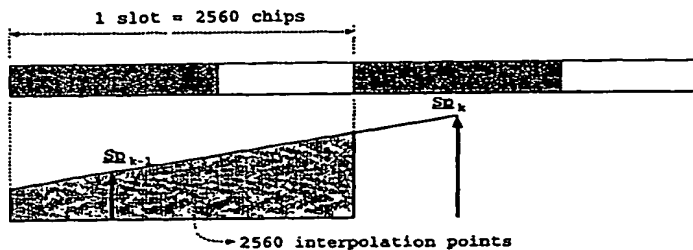


Figure 76: Interpolation for fast fading mode

The incoming Sp values are positioned in the middle of the pilot portion to calculate the other complex values. The pi\_po (pilot position) input is used for this. It is an integer in the range [0:2559]. In fig 76 pi\_po would be 768 or 769 ( $3/5 \cdot 2560/2$ ).

Linear interpolation is performed on both real and imaginary part of the Sp values. In this way we go via a straight line in the complex plane from Sp(k-1) to Sp(k).

$$Re[ces(i)] = (Re[Sp(k)] - Re[Sp(k-1)]) * (i - pi\_po)/2560 + Re[Sp(k-1)]$$

$$Im[ces(i)] = (Im[Sp(k)] - Im[Sp(k-1)]) * (i - pi\_po)/2560 + Im[Sp(k-1)]$$

with  $i = 0, 1, 2, \dots, 2559$  The 2560 different chips in a slot.

See fig 77 for an overview of the CCP finger process in case of channel mode 1.

The different pilot symbols are demodulated and coherently accumulated giving the values Sp0 to Sp5. The channel estimations ces(i) for the chips  $i$  of slot 2 are calculated during slot 4 with the aid of Sp2 and Sp3. So the Present and Future slot is used to make the channel estimates.

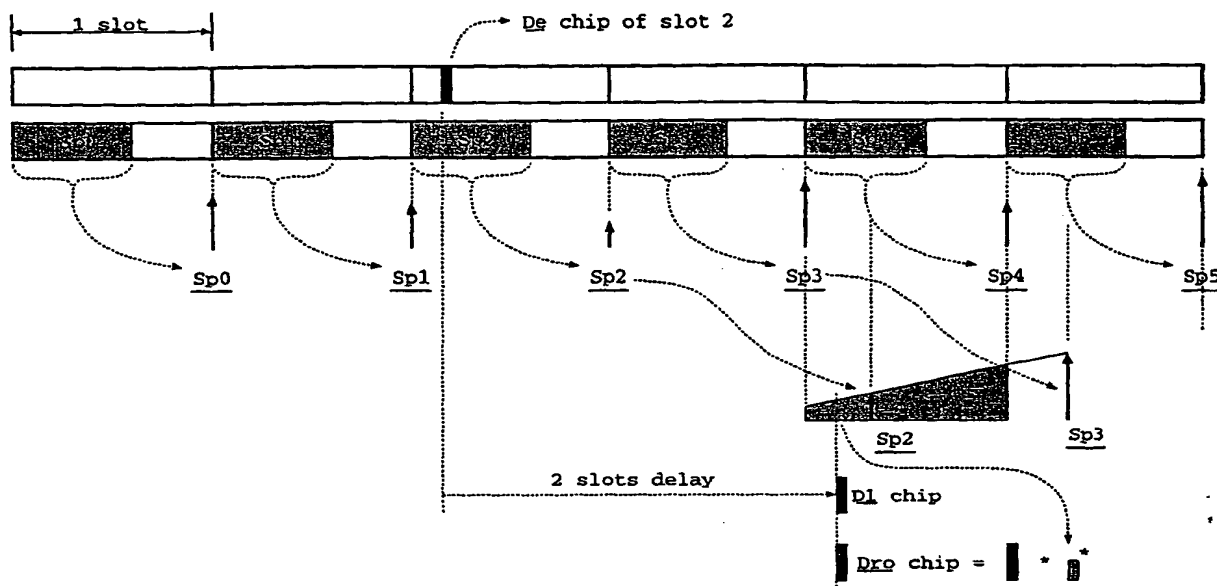


Figure 77: CCP finger process in the case of Fast fading mode

#### 14.1.7 Channel correction

This block has as input the delayed chips  $Dl$  coming from the FIFO and the channel estimations per chip  $ces$ . The function of this block is to correct for the channel phase of the finger and give a weight to the finger. The outputs from the different fingers can then be combined (coherently) in one signal. The following action is performed in these blocks :

$Dro = Dl * ces^*$  with  $ces^*$  the complex conjugates of  $ces$ .

#### 14.1.8 Zero forcing

Each finger output can be forced to zero with the  $zf$  signal.

zf,	C_CCP
0	$Dro$
1	$0 + j*0$

The purpose of this is to set a finger to 0 when no (or very little) signal is present in that finger to avoid the accumulation of a lot of noise.

The  $zf$  signal is obtained by comparing the slotwise FN and a programmable threshold.  $zf$  is 1 if  $FN \leq threshold$ .

## 15 Applications

!!!! 8-6-99 : Is not updated yet !!! First re-read UMTS docs !!!

## 15.1 UMTS uplink

### 15.1.1 Transmitter configuration

We have two physical channels, UL DPCH and PRACH with a different scrambling code. We could use set A for the DPCH, so one DPCCH and upto 7 DPDCH channels. Set B could be used for the PRACH. When PRACH and DPCH are exclusive channels (TBC) we are able to transmit 15 DPDCH channels.

### 15.1.2 Receiver configuration

TODO

## 15.2 UMTS downlink

For one MS with 2 DPDCH only.

### 15.2.1 Transmitter configuration

No info on synchronicity was found about the PSCCCH, as assumption we take that this is synchronous with the PDSCH channel.

SCH, PCCPCH, the starting phase of all downlink scrambling codes, PDSCH (TBC) and AICH access slots (TBC) are frame synchronous (PSCCCH assumed also).

Each DPCH and SCCPCH can have an individual frame timing with a resolution of 256 chips. (TBD) if it is meant within one downlink or between different downlinks. Wait for answer on mail from Pierre. Not so important as the resolution is 256 chips.

All channels are synchronous with a resolution of 256 chip (TBC).

SCH and PCCPCH are transmitted continuously. SCH are 2 QPSK channels, PCCPCH one QPN channel. SCCPCH are 2 exclusive (TBC) QPN channels (PCH and FACH). SCCPCH is exclusive with DL DPCH (TBC). Very little info on PDSCH, AICH and PSCCCH, we assume no exclusivity with any other channels. We also assume no multicode on this channels for the demonstrator with only one user. (TBC).

SCH could be mapped to set C.

PCCPCH, AICH, PDSCH and PSCCCH to set A.

2 DL DPCH and 2 SCCPCH could be mapped to set B, if synchronous (TBC).

If DL DPCH and SCCPCH are exclusive we could transmit upto 4 DL DPCHs.

### 15.2.2 Receiver configuration

TODO

## 16 Receiver wordlengths argumentation DRAFT

This section gives an argumentation for the used wordlengths in the receiver.  
Not always an exact theory but must be checked in simulation.

Used formula for implementation loss (see doc Lieven 15/6/99) :  
Implementation loss due to quantization noise  $IL = (N + QN)/N$

With  $N$  the noise present at a certain node,  $QN$  the inserted EXTRA quantization noise at that node.  $QN = 2^{-2b}/12$  with  $b$  the number of bits after the decimal point. This formula is only valid when going from inf bits to  $b$  bits and with a uniform distribution of the values. So  $IL$  is large for small  $N$ . So the worst case situation is for high  $S/N$ . So spread spectrum with low  $S/N$  is in favor.

In these calculations we will use a sine as signal with  $S/N = 20$  dB as worst case (pure QPSK). We assume that  $QN = 2^{-2b}/12$  is valid. We also assume that there is a safety margin of 2 bits at the MSB of the signal.

eg 8 bit ADC before decimal point, 2 MSB safety margin, gives sine amplitude of 32.

So  $S = 32^2/2 = 512$ , with  $S/N = 20$  dB  $\rightarrow N = 5.12$  with  $b = 2$  :  $IL = 10\log[(5.12 + 1/(12 \cdot 16))/5.12] = 0.004$  dB.

Warning  $IL$ s at successive nodes may not be added !!!

DC removal :

in  $s < 8, 0 >$ , out  $s < 10, 2 >$  (see above)

NCO :

cos, sin :  $s < 9, 8 >$  (see above, but we use full range (9bit) , no safety of 2 MSB  $\rightarrow$  si  $IL < 0.004$  dB which only uses 8 bit)

Downconv :

$s < 9, 8 > * s < 10, 2 > = s < 19, 10 >$  with 1 redundant MSB  $\rightarrow s < 18, 10 >$  addition of such 2 : FP output is  $s < 19, 10 >$ .

Same  $S$  and  $N$  in as out downconverter. So  $b = 2$  , with 1 extra MSB.  $\rightarrow$  cast downconv output to  $s < 11, 2 >$ .

We add one LSB = 1 to compensate truncation DC offset before going to the decimator.  $\rightarrow s < 12, 3 >$  Truncation energy would be : we truncate with  $b = 2$  , so DC offset of 0.125.  $\rightarrow 20\log(0.125) = -18$  dB. (Note : notes lugil 3/6/99  $\rightarrow$  with truncation with  $b = 0 \rightarrow$  we should get  $20\log(0.5) = -6$  dB. This is NOT what we get , in stead we have something of -4 dB. Every bit we add improves 6dB.

This is the only place in the RX where we do rounding !! (TBC).

Decimator :

In  $s < 12, 3 >$  , intern  $\rightarrow s < 22, 3 > \rightarrow$  level control needed. Level control again to  $s < 11, 2 >$  (0.004 dB). We probably don't have 2 safety MSBs here, so even better than 0.004 dB.

Filter :

49 taps

Coefficient number of bits :

$\rightarrow$  puts noise source on filter coefficients, total filter noise due to this effect is the multiplication of this noise source with the input data and added for all taps.

Worst case is when data is maximal. So we take ( $s < 11, 2 >$  in)  $DC = 2^8$  as data input.

2 extremes : NO correlation between the taps, and EVERY tap correlates with the other.



## 1. No correlation :

take  $b = 12 \rightarrow$  offset of coef  $= 2^{-12}/2$  $\rightarrow$  total  $QN = 49 * (2^8 * 2^{-12}/2)^2 = 0.048$ for  $S$  we take sine at  $1/4$  max amplitude in  $s_{11,2i} \rightarrow S = 2^{2*6}/2$ . To get an  $S/N$  of 20 dB  $\rightarrow N = 20.48$  $\rightarrow IL = 10 \log[(20.48 + 0.048)/20.48] = 0.01$  dB.

## 2. Complete correlation :

 $b = 12$  $\rightarrow QN = (49 * 2^8 * 2^{-12}/2)^2 \rightarrow$  same  $S \rightarrow IL = 0.5$  dB.

## 3. correlating in groups of 4.

 $\rightarrow QN = 12.25 * (4 * 2^8 * 2^{-12}/2)^2 \rightarrow$  same  $S \rightarrow IL = 0.04$  dB. $\rightarrow$  take 12 bits after decimal point, TBC by simulation. $4 < \text{sum}(\text{abs}(\text{coef})) < 8 \rightarrow 3$  MSB needed  $\rightarrow$  full precision filter output is  $s < 26, 14 >$ . This is reduced to  $s < 20, 8 >$  (so we have still 6 extra LSB compared with the input).

## Level control 2 :

output to 10 bits, this is more than AD number of bits, so certainly enough.

6045426-072299

---

# UTRA

Draft UMTS acquisition spec

Status: V1-a  
Doc no NL-UTRA-1  
July 12, 1999  
Nico Lugil



## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Abbreviations and conventions</b>	<b>4</b>
2.1	Abbreviations . . . . .	4
2.2	Conventions . . . . .	4
<b>3</b>	<b>General definitions</b>	<b>5</b>
<b>4</b>	<b>Synchronisation channel</b>	<b>6</b>
4.1	General synchronisation channel properties . . . . .	6
4.2	SCH TX hardware . . . . .	6
4.3	SCH RX hardware . . . . .	7
4.3.1	Initial cell search . . . . .	7
4.3.2	Idle mode cell search . . . . .	9
4.3.3	Active mode cell search . . . . .	9
<b>5</b>	<b>Physical Random Access channel</b>	<b>10</b>
5.1	General Physical Random Access channel properties . . . . .	10
5.2	PRACH TX hardware . . . . .	12
5.3	SCH RX hardware . . . . .	13

662220-92151109

## List of Figures

1	structure of SCH waveform . . . . .	6
2	SCH acquisition hardware for step 1 and 2 . . . . .	7
3	step 2 frame synchronisation and codegroup identification . . . . .	8
4	Access slots . . . . .	10
5	Structure of the random access burst . . . . .	10
6	message part structure . . . . .	11
7	TX PRACH overview . . . . .	12
8	Possible positions of the received preamble start . . . . .	14
9	Preamble acquisition hardware . . . . .	14

662220-924511-05

© 1998 by Sirius Communications NV. All rights reserved. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Sirius Communications NV. The information of this document is subject to change without notice. Sirius Communications NV shall not be responsible for any errors that may appear in this document. Sirius Communications NV makes no commitment to update or keep current the information contained in this document. Devices sold by Sirius Communications NV are covered by warranty and patent indemnification provisions appearing in Sirius Communications NV Terms and Conditions of Sale only.

50145426-072299

## 1 Introduction

This document is a collection of requirements for the chip to comply with the ETSI UTRA Submission document and the (received) XX updates.

This document contains the UMTS specification for each channel and conclusions for the chip hardware. For this text my simulation hardware models are used as reference hardware, implementation can differ. Note that all numbers given are for the 4.096 Mcps model. For the 8.192 and 16.384 Mcps modes all numbers are scaled (TBC). At the end of the document a more general transceiver model is described.

## 2 Abbreviations and conventions

### 2.1 Abbreviations

BS Base station  
MS Mobile station  
SF Spreading factor

### 2.2 Conventions

The following conventions apply in figures :

— real signal or 1 branch of a complex signal  
→ complex signal  
→ collection of signals (eg control signals)

1662260-072399

### 3 General definitions

Some definitions which apply for the branches of the channels in general:

- slot :  $0.625 \text{ ms}$  time period = 2560 chips = 10 to 64 symbols (SF 256 down to 4)
- frame :  $10 \text{ ms}$  time period = 16 slots
- superframe :  $720 \text{ ms}$  time period = 72 frames

666220" 92454109

## 4 Synchronisation channel

### 4.1 General synchronisation channel properties

- SCH
- downlink signal used by the MS for cell search (= synchronise with the BS)
- waveform :

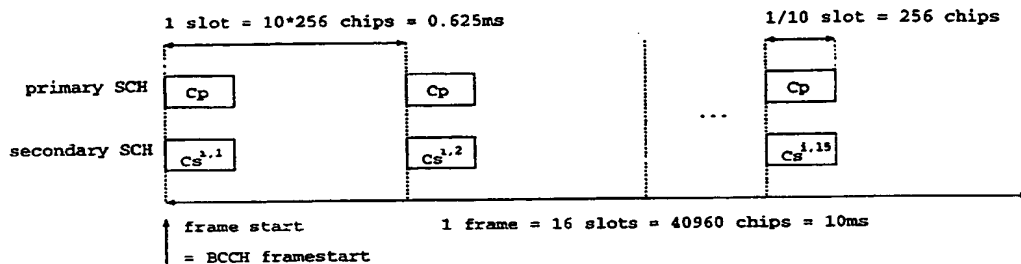


Figure 1: structure of SCH waveform

The SCH channel comprises two subchannels : the primary SCH (SCH1) and secondary SCH (SCH2).

SCH1 : unmodulated bursts (1/10) of Cp code, codelength 256. The start of the Cp transmission must be time aligned with the BCCH slot boundary.

SCH2 : consists of repeatedly transmitting a length 16 sequence of unmodulated codes of length 256 chips. These are transmitted parallel with the Cp code. The start of the SCH frame (so start of Cs1) must be time-aligned with the BCCH frame boundary (TBC).

- No scrambling
- Modulation : QPSK with on I and Q the same information (SCH1+SCH2) or SCH1 on I and Q on SCH2 (TBD). Cfr email Pierre - question 15.

### 4.2 SCH TX hardware

We could use a the general TX channels for the SCH channel (TBC). In this case there must be provisions to work in burst mode (eg usage of an activity bit) and it must be possible to 'switch off' the scrambler for some channels. (switch off but have the same delay to keep it synchronous with the other scrambled channels).

For the Cp code a 256 bit RAM

For the 16 Cs codes we could use one 256 bit RAM and update this code between each slot. Maximum time to update this RAM is (for the 16Mcps case)  $(625-62.5)/4 \text{ usec} = 140 \text{ usec}$ . TBC if this is enough.



### 4.3 SCH RX hardware

The cell search process is updated in XX.07, not available yet, so these thing are TBC.

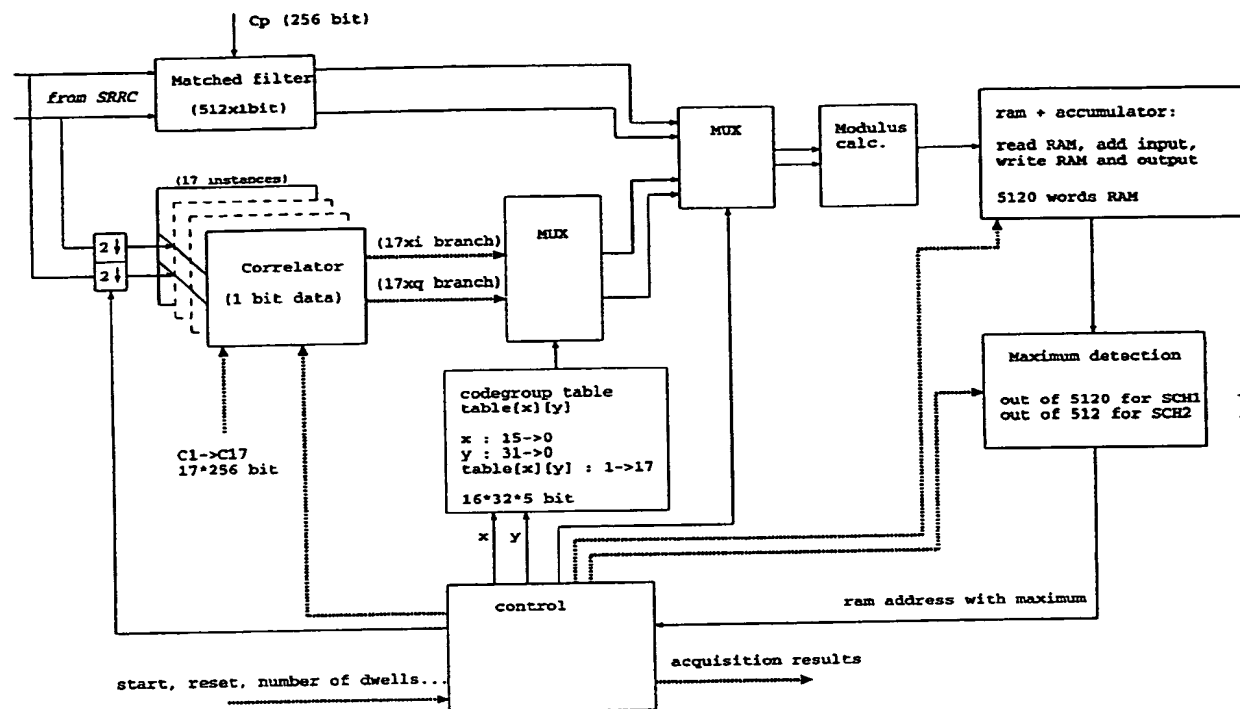


Figure 2: SCH acquisition hardware for step 1 and 2

3 modes are possible : initial cell search, idle mode cell search and active mode cell search.

#### 4.3.1 Initial cell search

The initial cell search consists of 3 steps :

##### Step 1 : Slot synchronisation

In this step, the MS acquires slot synchronisation by doing a fast acquisition on  $C_p$  which is common to all BSs. See figure 2.

The samples coming from the SRRC filter and at 2 fchip are sent through a matched filter with the  $C_p$  code. We use a matched filter with 1 bit input to avoid problems in selecting significant bits and offsets due to truncation. To compensate for this 2dB loss the matched filter uses half chip spacing in stead of 1 chip spacing. For each matched filter output the modulus (TBC) is calculated ( $\text{approx max}(\text{abs}(i,q) + 0.5 \text{ min}(\text{abs}(i,q)))$ ). So a coherent integration is done over 256 chips. The integration loss associated with this coherent integration and a maximum residual frequency of +/- 7kHz (TBC) is 2.93 dB. Spacing between slot edges at 2 fchip =  $2 \cdot 10 \cdot 256 = 5120$  half chips. The first 5120 moduli are written into the RAM. To get better reliability a non-coherent dwell is performed.

This is done by adding the next 5120 points to the previous 5120 points already present in the RAM. This is repeated for the required number of dwells. According to PR maximum 16 dwells will be needed. (with an  $E_b/N_0$  for SCH1 of around 10 db, this is equivalent with an SCH1 and 2 with a burstgain of 2 and 100 users with a gain of 1. Note  $E_b$  = burstpower = average power \* 10). If possible flexibility will be added to do more than 16 dwells. The effect of this will be most visible in the RAM wordsize (TBD).

During the last dwell, the maximum detection module finds the maximum in the last 5120 points written to the RAM, and returns the address of this value. This address identifies the slot edge.

The correlators can be switched off during this stage.

## Step 2 : Frame synchronisation and code-group identification

Each BS belongs to 1 of 32 possible codegroups. Each codegroup uses a different sequence of 16 Cs codes on SCH2. This length 16 sequence is a combination of 16 codes from a set of 17 secondary codewords (C1 → C17). Eg if the BS belongs to codegroup 1, the sent sequence is 'C1 C1 C2 C11 C6 C3 C15 C7 C8 C7 C15 C3 C6 C11 C2'

The data coming from the SRRC filter are subsampled by a factor 2 and with a phase determined by step1 (even or odd maximum address). From step1 the position of the secondary codes is known. During this burst 17 parallel 1 bit correlators correlate the subsampled data with each of the possible Cs codes (C1 → C17). The outputs of all the 17 correlators for 16 consecutive secondary SCH locations are used to FORM the decision variables of step2. The decision variables are obtained by non-coherently summing the correlator outputs corresponding to each 16 length sequence out of the 32 possible sequences and its 16 cyclic shifts. So from  $16 \times 17 = 272$  variables,  $32 \times 16 = 512$  decision variables are calculated. One decision variable is formed by adding 16 correlator outputs non-coherently (modulus). The calculation of these 512 values is distributed over the 16 idle intervals (9/10) after each correlation. See figure 3. The codegroup table in figure 2 contains the 32 different sequences of length 16. By generating the correct x and y (x = phaseshift, y = codegroup number) one of the 17 correlator outputs can be selected and the modulus is calculated. During the first slot, in the idle period, each of the codegroup table entries is selected, this selects 1 of the 17 correlator outputs and the modulus of these are written to one of the 512 used RAM positions during step2. During slot 2, each value in the RAM is added with 1 of the 17 correlator modulus outputs of slot2. Which correlator needs to be selected for each RAM address is selected by the x and y combination. This is repeated for the 16 slots, during slot 16 the resulting values are the 512 decision variables which are sent through the maximum detector. The address coming from the maximum detection block identifies the codegroup and the phase shift. The frame edge can be determined from the shift.

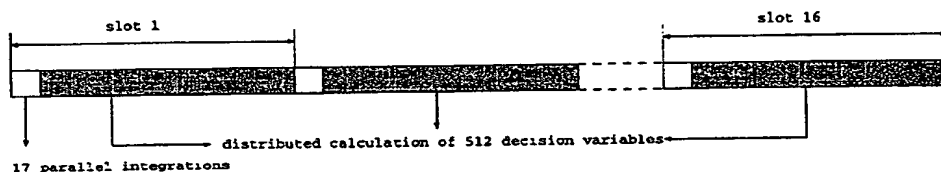


Figure 3: step 2 frame synchronisation and codegroup identification

At least 16 slots have to be added non-coherently. When in step1 the possibility is foreseen to use more than 16 dwells, the same number should be possible in step2 to get at least the same reliability. However in step2 it is best to use a multiple of 16 for the number of dwells. Again as in step1, more than 16 dwells are not required for UMTS mode (TBC).

## Step 3 : Scrambling code identification

Not yet implemented !!!!! (so not on figure)

In each codegroup (determined from step2) there are 16 possible scrambling codes used on the downlink. In this step this scrambling code will be determined. The start of the scrambling code is also known from step 2 (framestart TBC). The CCPCH (BCCH) channel is correlated symbol by symbol for each of the 16 possible scrambling codes. Different symbols are combined non-coherently. How many symbols have to be combined is TBD. This depends on the cross correlation between the 16 preamble codes, for each possible length. So best if this is kept programmable. If we should correlate over the complete scrambling code, this would mean a combination of  $16 \times 10 = 160$  symbols non-coherently.

TBD if the search over the 16 scrambling codes is done parallel (at the same time) or serial (each after another). Consider acquisition time (depending on how much symbols we use for this step) and needed hardware (requires extra descramblers).

After step3, BCCH information can be read.

Note that in the 3 steps almost 3 dB can be lost due to doppler and clock errors. After these steps do a carrier refinement with an FFT. In principle the FFT process can be done after step2 (TBC). Another proposition to 'avoid' the 3 dB loss is doing the 3 steps for different settings of the RX NCO. This should be done under software control.

#### 4.3.2 Idle mode cell search

TODO

#### 4.3.3 Active mode cell search

TODO

666220-92454109

## 5 Physical Random Access channel

### 5.1 General Physical Random Access channel properties

- PRACH

- Uplink channel to do a random access request.

- waveform :

The PRACH is based on a slotted ALOHA approach, i.e. a mobile station can start the transmission of the PRACH at a number of well-defined time-offsets relative to the frame boundary of the BCCH of the current cell. 8 different access slots are defined, spaced 1.25 ms as illustrated in fig 4.

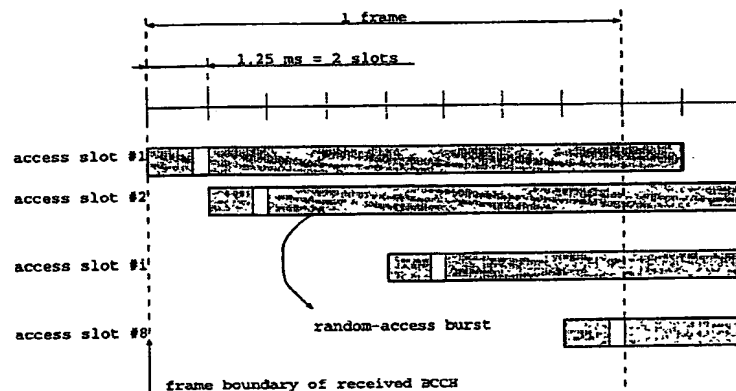


Figure 4: Access slots

The structure of the random-access burst is shown in fig 5.

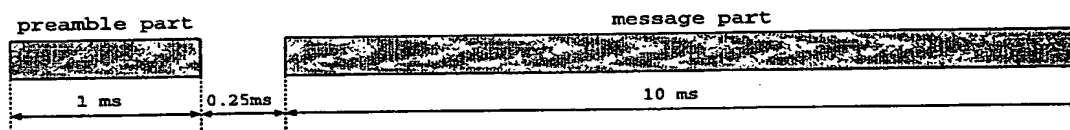


Figure 5: Structure of the random access burst

#### Preamble part :

The preamble part consists of a signature of length 16 complex symbols. Each preamble signal is spread with a 256 chip code. There are a total of 16 different signatures. The resulting complex data is QPSK modulated. Note that the complex symbols are  $(1+j)$  or  $(-1-j)$ . So we have a QPSK modulation with the same data on I and Q branch.

The preamble is not scrambled.

#### Idle time :

Between the preamble and message part is a 0.25 msec idle time. This 0.25 ms is preliminary. This makes for 4 idle symbols, spread with SF 256.

Message part :

The message part is a 10 msec (so a complete frame) burst. 2\*BPSK is used for this. See fig 6.

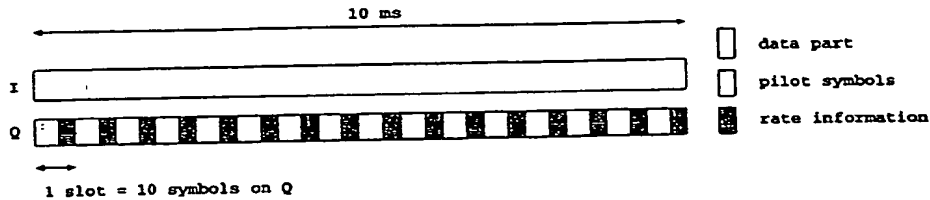


Figure 6: message part structure

The I branch is used to send the data part, the Q branch to send the control part. The control part is split in pilot symbols and rate information. The ratio between both is not fixed (TBD). The data part uses a SF of 256, 128, 64 or 32. The control part always uses SF 256. The rate information of the control part gives info about the data part SF. The data part ends with an 8 bit CRC to detect errors in the data part. Because of the different SF on I and Q, a different gain for I and Q branch must be foreseen after spreading. The spread and amplified complex signal is complex scrambled with a 40960 length scrambling code. Complex scrambling is the (complex) multiplication of 2 complex numbers: the complex spreaded data and the complex scrambling code.

662220-3044409

## 5.2 PRACH TX hardware

We could use one of the general TX channels to make the PRACH waveform (TBC). In that case the following things required for the general TX channels :

- An exact and easy way to work in burst mode (eg the use of an activity bit)
- The possibility to 'switch off' and 'switch on' the scramblers at an exact defined time. Note that when the scramblers are turned off, the data must still get the same delay as when the scramblers are turned on.
- An important requirement is that it must be possible to synchronise the PRACH with the received BCCH frame edge + an offset of  $(\text{access\_slot\_nr} - 1) * 1.25 \text{ ms}$ .

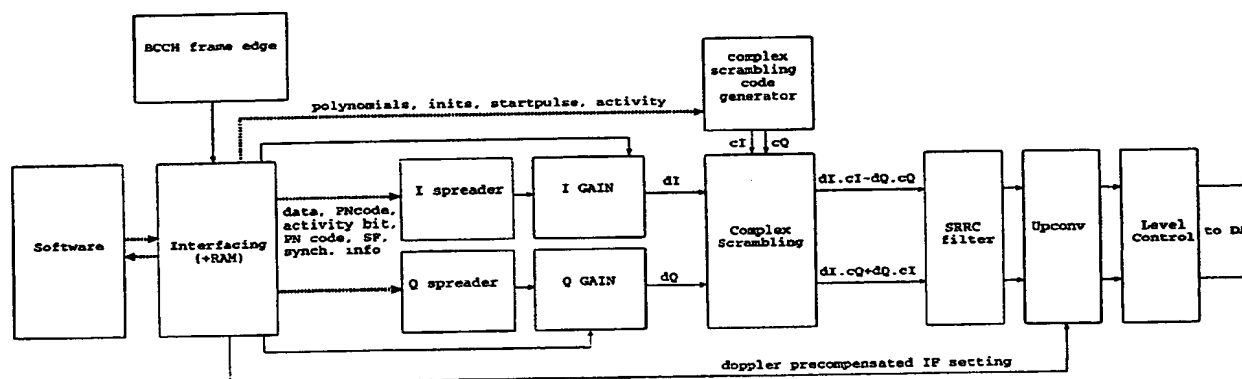


Figure 7: TX PRACH overview

### Software and interfacing:

The MS acquires synchronisation to a BS (SCH channel) and reads different parameters from the BCCH channel (preamble spreading codes, available signatures, available access slots, ...) (see UTRA submission document). The software decides which parameters to use, these parameters are passed to the hardware via the interface block. The start of the access slot offset must be synchronous with the BCCH frame edge. The upconverter must be set to a doppler precompensated IF in order to avoid great integration losses in the BS receiver.

### Spreaders:

The data coming from the interface block is spread with a max SF of 256. For the access slot offset and the idle period, the activity bit is set to 0. For each of the two branches 256bit RAM is needed to store the PN codes. TBC if the software+interfacing is fast enough to do the updates of the registers and RAM.

### Gain stage

The spreaded data is sent through a gain block. A different gain must be possible for I and Q branch.

### Complex scrambling code generator

Because this is a 40960 bit code, no RAM but a generator is needed here. The start of the generator must be controllable in order to synchronise with the start of the message part. The scrambling code is formed as follows (same scrambling codes as for the other dedicated uplink channels).

$$C_{scramb} = c_1(w_0 + jc_2'w_1)$$

where  $w_0$  and  $w_1$  are chip rate sequences defined as repetitions of:

$$w_0 = \{1\ 1\}$$

$$w_1 = \{1\ -1\}$$

and where  $c_1$  is a real chip rate code, and  $c_2'$  is a decimated version of the real chip rate code  $c_2$ . The preferred decimation factor is 2, however other decimation factors should be possible in future evolutions of UMTS if proved desirable.

With a decimation factor of  $N=2$ ,  $c_2'$  is given as :

$$c_2'(2k-1)=c_2'(2k)=c_2(2k-1), k=1,2,3...$$

$c_1$  and  $c_2$  are constructed as the position wise modulo 2 sum of 40960 chip segment of two binary m-sequences generated by means of two generator polynomials of degree 41. These codes thus have a period of one radio frame of 10ms.

The code  $c_2$ , used in generating the quadrature component of the complex spreading code is a 1024-chip shifted version of the code  $c_1$  used in generating the in phase component. TBC because I think this conflicts with Cscramb =  $c_1(w_0 + jc_2'w_1)$ .

The code generator must be able to generate the sequence shifted arbitrarily from the initial state. (TBD why ?)

*For other channels or applications other possibilities should be foreseen:*

*Possibility to use  $c_1$  and  $c_2$  directly as the real and imaginary part of Cscramb, so  $Cscramb = c_1 + j*c_2$*

*Possibility to re-initialise the gold code generators at any given time, so this must not coincide with a frame edge.*

*For example you could just let the generators run freely from a given start moment.*

### Complex scrambling

In this block a complex scrambling is performed:  $(dI+jdQ)*(cI+jcQ) = dI.cI-dQ.cQ + j(dI.cQ+dQ.cI)$

There must be the possibility to 'switch off' and 'switch on' the scramblers at an exact defined time. Note that when the scramblers are turned off, the data must still get the same delay as when the scramblers are turned on.

*For other channels or applications other possibilities should be foreseen: No complex scrambling, but 2 times a real scrambling : In stead of  $(dI+jdQ)*(cI+jcQ) \rightarrow dI*cI + j\ dQ*cQ$ .*

## 5.3 SCH RX hardware

Incomplete, only the preamble acquisition is considered here.

### Preamble acquisition

In a first step, the BS will search for a preamble signature and determine its timing. So 3 things that have to be performed more or less at the same time : judge if a preamble signature is present, find its timing and see which signature was used. The chip will be restricted in the way that preamble acquisition will only be done with 1 of the 16 possible preamble signatures possible. The maximum round trip delay is smaller than one bit of the signature (TBC certainly for the 16.384 Mcps case). To comply with this, the maximum distance between MS and BS should be :

for the 4.096 Mcps case : max distance BS-MS  $< 0.5 * (3 * 10^8 m/sec * 1/16ms) = 9.3km$

for the 16.384 Mcps case : max distance BS-MS  $< 0.5 * (3 * 10^8 m/sec * 0.25/16ms) = 2.3km$

In this case the chipphase search space for the preamble signature start is 1 symbol of 256 chips, starting from the BCCH frame edge plus each of the 8 possible offsets. See fig 8

Note : This figure gives the possible locations of the preamble start (=first chip) AT THE INPUT OF THE RECEIVER. So it is very important to keep in mind the delays of the RX (eg the matched filter has a delay of 1 symbol) when slaving to the BCCH !!!! Note that in this calculations the TX and RX chain delays were neglected.

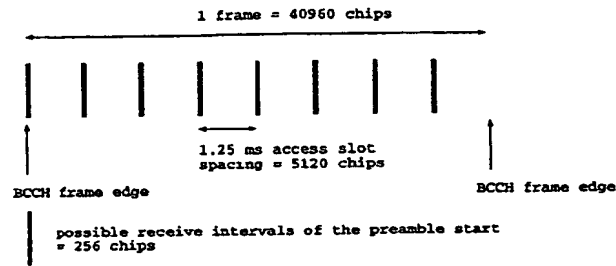


Figure 8: Possible positions of the received preamble start

This reduces the cell radius further. In principle this could be compensated in the BS receiver by not using the transmit frame edge of the BCCH as a reference for the search space, but the BCCH frame edge + TX and RX chains delays (TBD if this is necessary).

From fig 4 it is clear that the different possible locations of the preamble + 1 bit do not overlap. So no parallel searches between different access slots are needed. See fig 9 for preamble acquisition hardware.

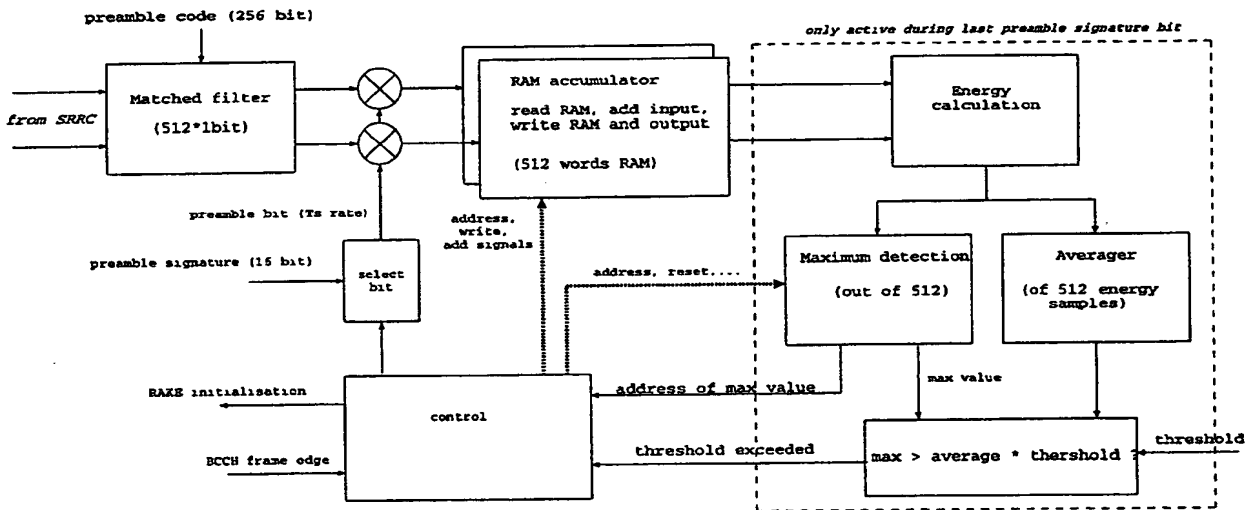


Figure 9: Preamble acquisition hardware

The acquisition process starts at the start of, 1 of the 8 possible, locations of the possible reception of the first preamble bit. The signal coming from the SRRC filters is fed to a matched filter with the preamble spreading code, working at 2 fchip (half chip spacing) and working with a 1 bit data input. The first 512 complex values coming from the matched filter are multiplied with the first preamble bit of the preamble signature and written in a complex RAM block. So a separate RAM for the real and imaginary part (or 1 RAM with double wordsize). The next 512 complex points coming from the matched filter are multiplied with the second preamble bit of the preamble signature and added to the first 512 complex points already in the RAM. This is done for the complete length of





July 12, 1999  
Revision 1-a

the preamble signature (16 bits). In this way we get a coherent integration over the complete preamble signature (=1ms) and a processing gain of 36 dB. The last 512 points accumulated in the RAM (so during preamble bit 16) are sent to the energy block where the energy is calculated for each of the 512 complex values. The maximum of these 512 results is sought and the average of these 512 values is calculated. The maximum value is compared with the average times a threshold value. If the maximum is larger than this, the address of the maximum value is sent to the control part, which from the address can determine the timing of the received PRACH. The control part then initiates the RAKE and goes to the second stage : detection of the message part. If the threshold was not exceeded, the process restarts at the next possible location of the possible reception of the first preamble bit. With the large coherent integration time of 1 ms and a maximum doppler of  $\pm 7$  kHz (TBC), it is absolutely necessary to perform doppler precompensation in the mobile station to avoid an enormous integration loss.

**SECRET**

**Detection of the message part**

**TODO**

some points already :

CRC at end message part to reject false detections of preamble.

662270-92454709

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☒ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER: \_\_\_\_\_**

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**

**This Page Blank (uspto)**